(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号
特開2003－8445
（P2003－8445A）
(43)公開日　平成15年１月10日(2003.1.10)

(51)Int.Cl.[7]	識別記号	ＦＩ		テーマコード（参考）
Ｈ０３Ｍ	7/30	Ｈ０３Ｍ	7/30	Ａ　５Ｃ０５９
	7/40		7/40	５Ｃ０７８
	7/46		7/46	５Ｊ０６４
Ｈ０４Ｎ	1/41	Ｈ０４Ｎ	1/41	Ｚ
	7/30		7/133	Ｚ

審査請求　未請求　請求項の数30　ＯＬ　（全 50 頁）

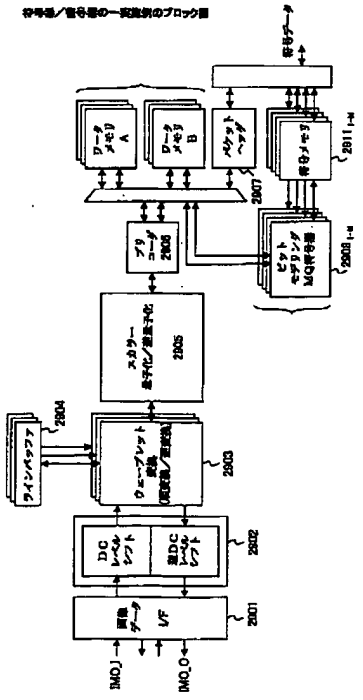
(21)出願番号　特願2002－95109(P2002－95109)
(22)出願日　平成14年３月29日(2002.3.29)
(31)優先権主張番号　８２３７３３
(32)優先日　平成13年３月30日(2001.3.30)
(33)優先権主張国　米国（ＵＳ）

(71)出願人　000006747
株式会社リコー
東京都大田区中馬込１丁目３番６号
(72)発明者　エドワード　エル．シュワルツ
アメリカ合衆国，カリフォルニア　94025，メンロ・パーク，サンド・ヒル・ロード　2882番，スイート　115　リコーイノベーション内
(74)代理人　100070150
弁理士　伊東　忠彦

最終頁に続く

(54)【発明の名称】　コンテキストモデルによるラン・スキップカウントに基づくメモリアクセス及びスキッピング

(57)【要約】
【課題】　情報を符号化及び復号化する方法並びに装置を提供する。
【解決手段】　本発明の装置は、メモリ及び復号化ハードウェアを含む。メモリは、ランカウント及び／又はスキップカウントを保持し、復号化ハードウェアは、復号化中にメモリから取得されたランカウント及び／又はスキップカウントを復号化する。



1

【特許請求の範囲】

【請求項１】　係数のグループに関して有意プロパゲーション・パスを実行する手順と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する手順と、
リファインメント・サブビットプレーン・パスを実行する手順と、を有し、
リファインメント・サブビットプレーン・パスは、
リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスする手順と、
リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手順と、
メモリから取得したリファインメント・ビットを符号化する手順と、によって実行される、方法。

【請求項２】　複数のリファインメント・ビットが取得され、非リファインメント・ビットは無視される、請求項１記載の方法。

【請求項３】　係数のグループは符号ブロックを構成する、請求項１記載の方法。

【請求項４】　上記データ構造を作成する手順は、係数のグループに対しリファインメント・サブビットプレーン・パスの係数の場所を記述するデータ構造を作成することにより、少なくとも一つのデータ構造を作成する、請求項１記載の方法。

【請求項５】　有意プロパゲーション・パス中に、第１のデータ構造はリファインメント・ビットのため作成され、第２のデータ構造はクリーンアップ・ビットのため作成される、請求項４記載の方法。

【請求項６】　データ構造は、リファインメント・ビットの各ランの標識と、リファインメント・ビットの次のランの前に係数のグループ内でスキップすべき係数の個数と、を含む、請求項４記載の方法。

【請求項７】　リファインメント・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、
カウント1は符号語000、
カウント2は符号語001、
カウント3は符号語010、
カウント4は符号語011、
カウント5は符号語1000000000000、
カウント6は符号語1000000000001、以下同様に、
カウント4096は符号語1111111111011のような可変長符号で表現される、請求項６記載の方法。

【請求項８】　リファインメント・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、
カウント1は符号語0、

2

カウント2は符号語01、
カウント3は符号語1100、
カウント4は符号語1101_0、
カウント5は符号語1101_1、
カウント6は符号語1110_0000、
カウント7は符号語1110_0001、同様に、
カウント21は符号語1110_1111、
カウント22は符号語1111_0000_0000_0000、
カウント23は符号語1111_0000_0000_0001、以下同様に、カウント4096は符号語1111_1111_1110_1010のような可変長符号で表現される、請求項６記載の方法。

【請求項９】　リファインメント・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、
カウント1は、γ' フォーマットの符号語0とγフォーマットの符号語0、
カウント2は、γ' フォーマットの符号語10_0とγフォーマットの符号語100、
カウント3は、γ' フォーマットの符号語10_1とγフォーマットの符号語110、
カウント4は、γ' フォーマットの符号語110_00とγフォーマットの符号語10100、
カウント5は、γ' フォーマットの符号語110_01とγフォーマットの符号語10110、
カウント6は、γ' フォーマットの符号語110_10とγフォーマットの符号語11100、
カウント7は、γ' フォーマットの符号語110_11とγフォーマットの符号語11110、
カウント8は、γ' フォーマットの符号語1110_000とγフォーマットの符号語1010100、
カウント9は、γ' フォーマットの符号語1110_001とγフォーマットの符号語1010110、以下同様に、
カウント15は、γ' フォーマットの符号語1110_111とγフォーマットの符号語1111110、
カウント16は、γ' フォーマットの符号語11110_0000とγフォーマットの符号語101010100、
カウント32は、γ' フォーマットの符号語111110_00000とγフォーマットの符号語10101010100、
カウント64は、γ' フォーマットの符号語111110_000000とγフォーマットの符号語1010101010100、
カウント128は、γ' フォーマットの符号語11111110_0000000とγフォーマットの符号語101010101010100、
カウント256は、γ' フォーマットの符号語111111110_00000000とγフォーマットの符号語1010101010101010100、
カウント512は、γ' フォーマットの符号語1111111110_000000000とγフォーマットの符号語10101010101010101000、
カウント1024は、γ' フォーマットの符号語11111111110_000000000とγフォーマットの符号語1010101010101010100、

カウント2048は、γ' フォーマットの符号語1111111111110_0000000000とγフォーマットの符号語10101010101010101010100、
カウント4096は、γ' フォーマットの符号語1111111111110_00000000000とγフォーマットの符号語1010101010101010101010100、のようなガンマ符号で表現される、請求項６記載の方法。

【請求項１０】　リファインメント・ビットの各ランの標識は整数として保持される、請求項１記載の方法。

【請求項１１】　整数は最小限のサイズの整数で表わされる、請求項１０記載の方法。

【請求項１２】　スキップすべき係数の個数は整数として保持される、請求項１記載の方法。

【請求項１３】　整数は最小限のサイズの整数で表わされる、請求項１２記載の方法。

【請求項１４】リファインメント・ビットの各ランの標識、及び、スキップすべき係数の個数は整数として保持される、請求項１記載の方法。

【請求項１５】　整数は最小限のサイズの整数で表わされる、請求項１４記載の方法。

【請求項１６】　係数のグループに関して有意プロパゲーション・パスを実行する手段と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する手段と、
リファインメント・サブビットプレーン・パスを実行する手段と、
を有し、
リファインメント・サブビットプレーン・パスは、
リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスする手段と、
リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手段と、
メモリから取得したリファインメント・ビットを符号化する手段と、によって実行される、装置。

【請求項１７】　複数のリファインメント・ビットが取得され、非リファインメント・ビットは無視される、請求項１６記載の装置。

【請求項１８】　係数のグループは符号ブロックを構成する、請求項１６記載の装置。

【請求項１９】　上記データ構造を作成する手段は、係数のグループに対しリファインメント・サブビットプレーン・パスの係数の場所を記述するデータ構造を作成することにより、少なくとも一つのデータ構造を作成する手段を有する、請求項１６記載の装置。

【請求項２０】　有意プロパゲーション・パス中に、第１のデータ構造はリファインメント・ビットのため作成され、第２のデータ構造はクリーンアップ・ビットのため作成される、請求項１９記載の装置。

【請求項２１】　データ構造は、リファインメント・ビットの各ランの標識と、リファインメント・ビットの次のランの前に係数のグループ内でスキップすべき係数の個数と、を含む、請求項１９記載の装置。

【請求項２２】　リファインメント・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、
カウント1は符号語000、
カウント2は符号語001、
カウント3は符号語010、
カウント4は符号語011、
カウント5は符号語1000000000000、
カウント6は符号語1000000000001、以下同様に、
カウント4096は符号語1111111111011のような可変長符号で表現される、請求項２１記載の装置。

【請求項２３】　リファインメント・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、
カウント1は符号語0、
カウント2は符号語01、
カウント3は符号語1100、
カウント4は符号語1101_0、
カウント5は符号語1101_1、
カウント6は符号語1110_0000、
カウント7は符号語1110_0001、同様に、
カウント21は符号語1110_1111、
カウント22は符号語1111_0000_0000_0000、
カウント23は符号語1111_0000_0000_0001、以下同様に、
カウント4096は符号語1111_1111_1110_1010
のような可変長符号で表現される、請求項２１記載の装置。

【請求項２４】　リファインメント・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、
カウント1は、γ' フォーマットの符号語0とγフォーマットの符号語0、
カウント2は、γ' フォーマットの符号語10_0とγフォーマットの符号語100、
カウント3は、γ' フォーマットの符号語10_1とγフォーマットの符号語110、
カウント4は、γ' フォーマットの符号語110_00とγフォーマットの符号語10100、
カウント5は、γ' フォーマットの符号語110_01とγフォーマットの符号語10110、
カウント6は、γ' フォーマットの符号語110_10とγフォーマットの符号語11100、
カウント7は、γ' フォーマットの符号語110_11とγフォーマットの符号語11110、

カウント8は、γ' フォーマットの符号語1110_000とγフォーマットの符号語1010100、
カウント9は、γ' フォーマットの符号語1110_001とγフォーマットの符号語1010110、以下同様に、
カウント15は、γ' フォーマットの符号語1110_111とγフォーマットの符号語1111110、
カウント16は、γ' フォーマットの符号語11110_0000とγフォーマットの符号語101010100、
カウント32は、γ' フォーマットの符号語111110_00000とγフォーマットの符号語10101010100、
カウント64は、γ' フォーマットの符号語111110_000000とγフォーマットの符号語1010101010100、
カウント128は、γ' フォーマットの符号語11111110_0000000とγフォーマットの符号語101010101010100、
カウント256は、γ' フォーマットの符号語111111110_00000000とγフォーマットの符号語10101010101010100、
カウント512は、γ' フォーマットの符号語1111111110_000000000とγフォーマットの符号語1010101010101010100、
カウント1024は、γ' フォーマットの符号語11111111110_000000000とγフォーマットの符号語1010101010101010100、
カウント2048は、γ' フォーマットの符号語11111111110_0000000000とγフォーマットの符号語10101010101010101010100、
カウント4096は、γ' フォーマットの符号語11111111110_00000000000とγフォーマットの符号語1010101010101010101010100、のようなガンマ符号で表現される、請求項２１記載の装置。

【請求項２５】　係数のグループに関して有意プロパゲーション・パスを実行する機能と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する機能と、
リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスし、リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスし、メモリから取得したリファインメント・ビットを符号化することによって、リファインメント・サブビットプレーン・パスを実行する機能と、をコンピュータに実現させるためのプログラム。

【請求項２６】　係数のグループに関して有意プロパゲーション・パスを実行する手順と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する手順と、
クリーンアップ・サブビットプレーン・パスを実行する手順と、を有し、
クリーンアップ・サブビットプレーン・パスは、
クリーンアップ・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスする手順と、
クリーンアップ・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手順と、
メモリから取得したクリーンアップ・ビットを符号化する手順と、によって実行される、方法。

【請求項２７】　複数のクリーンアップ・ビットが取得され、非クリーンアップ・ビットは無視される、請求項２６記載の方法。

【請求項２８】　係数のグループは符号ブロックを構成する、請求項２６記載の方法。

【請求項２９】　上記データ構造を作成する手順は、係数のグループに対しクリーンアップ・サブビットプレーン・パスの係数の場所を記述するデータ構造を作成することにより、少なくとも一つのデータ構造を作成する、請求項２６記載の方法。

【請求項３０】　データ構造は、クリーンアップ・ビットの各ランの標識と、クリーンアップ・ビットの次のランの前に係数のグループ内でスキップすべき係数の個数と、を含む、請求項２９記載の方法。

【請求項３１】　クリーンアップ・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、
カウント1は符号語000、
カウント2は符号語001、
カウント3は符号語010、
カウント4は符号語011、
カウント5は符号語1000000000000、
カウント6は符号語1000000000001、以下同様に、
カウント4096は符号語1111111111011のような可変長符号で表現される、請求項３０記載の方法。

【請求項３２】　クリーンアップ・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、
カウント1は符号語0、
カウント2は符号語01、
カウント3は符号語1100、
カウント4は符号語1101_0、
カウント5は符号語1101_1、
カウント6は符号語1110_0000、
カウント7は符号語1110_0001、同様に、
カウント21は符号語1110_1111、
カウント22は符号語1111_0000_0000_0000、
カウント23は符号語1111_0000_0000_0001、以下同様に、
カウント4096は符号語1111_1111_1110_1010のような可変長符号で表現される、請求項３０記載の方法。

【請求項３３】　クリーンアップ・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方

は、
カウント1は、γ' フォーマットの符号語0とγフォーマットの符号語0、
カウント2は、γ' フォーマットの符号語10_0とγフォーマットの符号語100、
カウント3は、γ' フォーマットの符号語10_1とγフォーマットの符号語110、
カウント4は、γ' フォーマットの符号語110_00とγフォーマットの符号語10100、
カウント5は、γ' フォーマットの符号語110_01とγフォーマットの符号語10110、
カウント6は、γ' フォーマットの符号語110_10とγフォーマットの符号語11100、
カウント7は、γ' フォーマットの符号語110_11とγフォーマットの符号語11110、
カウント8は、γ' フォーマットの符号語1110_000とγフォーマットの符号語1010100、
カウント9は、γ' フォーマットの符号語1110_001とγフォーマットの符号語1010110、以下同様に、
カウント15は、γ' フォーマットの符号語1110_111とγフォーマットの符号語1111110、
カウント16は、γ' フォーマットの符号語11110_0000とγフォーマットの符号語101010100、
カウント32は、γ' フォーマットの符号語111110_00000とγフォーマットの符号語10101010100、
カウント64は、γ' フォーマットの符号語111110_000000とγフォーマットの符号語1010101010100、
カウント128は、γ' フォーマットの符号語11111110_0000000とγフォーマットの符号語101010101010100、
カウント256は、γ' フォーマットの符号語111111110_00000000とγフォーマットの符号語10101010101010100、
カウント512は、γ' フォーマットの符号語1111111110_000000000とγフォーマットの符号語1010101010101010100、
カウント1024は、γ' フォーマットの符号語11111111110_000000000とγフォーマットの符号語10101010101010101010100、
カウント2048は、γ' フォーマットの符号語11111111110_0000000000とγフォーマットの符号語101010101010101010101010100、
カウント4096は、γ' フォーマットの符号語11111111110_00000000000とγフォーマットの符号語1010101010101010101010101010100、のようなガンマ符号で表現される、請求項３０記載の方法。

【請求項３４】　クリーンアップ・ビットの各ランの標識は可変長符号で表現される、請求項３０記載の方法。

【請求項３５】　スキップすべき係数の個数は可変長符号で表現される、請求項３０記載の方法。

【請求項３６】　クリーンアップ・ビットの各ランの標識、及び、スキップすべき係数の個数は可変長符号で表現される、請求項３０記載の方法。

【請求項３７】　クリーンアップ・ビットの各ランの標識は整数として保持される、請求項３０記載の方法。

【請求項３８】　整数は最小限のサイズの整数で表わされる、請求項３７記載の方法。

【請求項３９】　スキップすべき係数の個数は整数として保持される、請求項３０記載の方法。

【請求項４０】　整数は最小限のサイズの整数で表わされる、請求項３９記載の方法。

【請求項４１】クリーンアップ・ビットの各ランの標識、及び、スキップすべき係数の個数は整数として保持される、請求項３０記載の方法。

【請求項４２】　整数は最小限のサイズの整数で表わされる、請求項４１記載の方法。

【請求項４３】　データ構造を作成する手順は、係数のグループに対するリファインメント・サブビットプレーン・パスの係数の場所を記述する第１のデータ構造、及び、係数のグループに対するクリーンアップ・サブビットプレーン・パスにの係数の場所を記述する第２のデータ構造を作成することにより、少なくとも一つのデータ構造を作成する、請求項２６記載の方法。

【請求項４４】　係数のグループは符号ブロックを構成する、請求項４３記載の方法。

【請求項４５】　係数のグループに関して有意プロパゲーション・パスを実行する手段と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する手段と、
クリーンアップ・サブビットプレーン・パスを実行する手段と、を有し、
クリーンアップ・サブビットプレーン・パスは、
クリーンアップ・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスする手段と、
クリーンアップ・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手段と、
メモリから取得したクリーンアップ・ビットを符号化する手段と、によって実行される、
装置。

【請求項４６】　複数のクリーンアップ・ビットが取得され、非クリーンアップ・ビットは無視される、請求項４５記載の装置。

【請求項４７】　係数のグループは符号ブロックを構成する、請求項４５記載の装置。

【請求項４８】　上記データ構造を作成する手段は、係数のグループに対しクリーンアップ・サブビットプレーン・パスの係数の場所を記述するデータ構造を作成することにより、少なくとも一つのデータ構造を作成する手段を有する、請求項４５記載の装置。

【請求項４９】　データ構造は、クリーンアップ・ビッ

トの各ランの標識と、クリーンアップ・ビットの次のランの前に係数のグループ内でスキップすべき係数の個数と、を含む、請求項４８記載の装置。

【請求項５０】　クリーンアップ・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、

カウント1は符号語000、

カウント2は符号語001、

カウント3は符号語010、

カウント4は符号語011、

カウント5は符号語1000000000000、

カウント6は符号語1000000000001、以下同様に、

カウント4096は符号語1111111111011のような可変長符号で表現される、請求項４９記載の装置。

【請求項５１】　クリーンアップ・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、

カウント1は符号語0、

カウント2は符号語01、

カウント3は符号語1100、

カウント4は符号語1101_0、

カウント5は符号語1101_1、

カウント6は符号語1110_0000、

カウント7は符号語1110_0001、同様に、

カウント21は符号語1110_1111、

カウント22は符号語1111_0000_0000_0000、

カウント23は符号語1111_0000_0000_0001、以下同様に、

カウント4096は符号語1111_1111_1110_1010のような可変長符号で表現される、請求項４９記載の装置。

【請求項５２】　クリーンアップ・ビットの各ランの標識とスキップすべき係数の個数のうちの少なくとも一方は、

カウント1は、γ' フォーマットの符号語0とγフォーマットの符号語0、

カウント2は、γ' フォーマットの符号語10_0とγフォーマットの符号語100、

カウント3は、γ' フォーマットの符号語10_1とγフォーマットの符号語110、

カウント4は、γ' フォーマットの符号語110_00とγフォーマットの符号語10100、

カウント5は、γ' フォーマットの符号語110_01とγフォーマットの符号語10110、

カウント6は、γ' フォーマットの符号語110_10とγフォーマットの符号語11100、

カウント7は、γ' フォーマットの符号語110_11とγフォーマットの符号語11110、

カウント8は、γ' フォーマットの符号語1110_000とγフォーマットの符号語1010100、

カウント9は、γ' フォーマットの符号語1110_001とγフォーマットの符号語1010110、以下同様に、

カウント15は、γ' フォーマットの符号語1110_111とγフォーマットの符号語1111110、

カウント16は、γ' フォーマットの符号語11110_0000とγフォーマットの符号語101010100、

カウント32は、γ' フォーマットの符号語111110_00000とγフォーマットの符号語10101010100、

カウント64は、γ' フォーマットの符号語111110_000000とγフォーマットの符号語1010101010100、

カウント128は、γ' フォーマットの符号語11111110_0000000とγフォーマットの符号語101010101010100、

カウント256は、γ' フォーマットの符号語111111110_00000000とγフォーマットの符号語10101010101010100、

カウント512は、γ' フォーマットの符号語1111111110_000000000とγフォーマットの符号語1010101010101010100、

カウント1024は、γ' フォーマットの符号語11111111110_000000000とγフォーマットの符号語101010101010101010100、

カウント2048は、γ' フォーマットの符号語11111111110_0000000000とγフォーマットの符号語1010101010101010101010100、

カウント4096は、γ' フォーマットの符号語11111111110_00000000000とγフォーマットの符号語1010101010101010101010100、のようなガンマ符号で表現される、請求項４９記載の装置。

【請求項５３】　クリーンアップ・ビットの各ランの標識は可変長符号で表現される、請求項４９記載の方法。

【請求項５４】　スキップすべき係数の個数は可変長符号で表現される、請求項４９記載の方法。

【請求項５５】　クリーンアップ・ビットの各ランの標識、及び、スキップすべき係数の個数は可変長符号で表現される、請求項４９記載の方法。

【請求項５６】　係数のグループに関して有意プロパゲーション・パスを実行する機能と、

後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する機能と、

クリーンアップ・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスし、クリーンアップ・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスし、メモリから取得したクリーンアップ・ビットを符号化することによって、クリーンアップ・サブビットプレーン・パスを実行する機能と、をコンピュータに実現させるためのプログラム。

【請求項５７】　リファインメント・ビットのランのサイズを指定するリファインメント・ビットランカウント、リファインメント・ビットのランの間でスキップすべき係数の個数を指定するリファインメント・ビットス

キップカウント、クリーンアップ・ビットのランのサイズを指定するクリーンアップ・ビットランカウント、及び、クリーンアップ・ビットのランの間でスキップすべき係数の個数を指定するクリーンアップ・ビットスキップカウントを収容し、リファインメント・パス又はクリーンアップ・パスの間に処理されるべき係数のグループ内での係数の場所を指示する、少なくとも一つのデータ構造を、コンテキストモデルが読み出す手順と、
コンテキストモデルが、少なくとも一つのデータ構造内の情報に基づいてメモリにアクセスする手順と、を有する方法。

【請求項５８】　メモリと、メモリに接続されたコンテキストモデルとを具備し、
コンテキストモデルは、リファインメント・ビットのランのサイズを指定するリファインメント・ビットランカウント、リファインメント・ビットのランの間でスキップすべき係数の個数を指定するリファインメント・ビットスキップカウント、クリーンアップ・ビットのランのサイズを指定するクリーンアップ・ビットランカウント、及び、クリーンアップ・ビットのランの間でスキップすべき係数の個数を指定するクリーンアップ・ビットスキップカウントを収容し、リファインメント・パス又はクリーンアップ・パスの間に処理されるべき係数のグループ内での係数の場所を指示する、少なくとも一つのデータ構造を読み出し、
コンテキストモデルは、少なくとも一つのデータ構造内の情報に基づいてメモリにアクセスする、復号器。

【請求項５９】　リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するため、後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造にアクセスする手順と、
リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手順と、
メモリから取得したリファインメント・ビットを符号化する手順と、を有する方法。

【請求項６０】　上記データ構造を作成すると共に、有意プロパゲーション・パスの係数を符号化することにより、有意プロパゲーション・パスを実行する手順を更に有する請求項５９記載の方法。

【請求項６１】　リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するため、後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造にアクセスする手段と、
リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手段と、
メモリから取得したリファインメント・ビットを符号化する手段と、を有する装置。

【請求項６２】　上記データ構造を作成すると共に、有意プロパゲーション・パスの係数を符号化する手段を含み、有意プロパゲーション・パスを実行する手段を更に有する請求項５９記載の装置。

【請求項６３】　クリーンアップ・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するため、後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造にアクセスする手順と、
クリーンアップ・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手順と、
メモリから取得したクリーンアップ・ビットを符号化する手順と、を有する方法。

【請求項６４】　上記データ構造を作成すると共に、有意プロパゲーション・パスの係数を符号化することにより、有意プロパゲーション・パスを実行する手順を更に有する請求項６３記載の方法。

【請求項６５】　ランカウントを保持する第１の部分、スキップカウントを保持する第２の部分、及び、第１の部分と第２の部分を分離する第３の部分を有するメモリと、
上記メモリに接続され、上記メモリから同時に獲得されたランカウント及びスキップカウントを復号化する復号器と、を具備した情報を復号化する装置。

【請求項６６】　上記メモリの第３の部分は、上記第１の部分と上記第２の部分の間のメモリの使用されない部分を含む、請求項６５記載の装置。

【請求項６７】　上記メモリの使用されない部分は、上記第１の部分及び上記第２の部分に隣接している、請求項６６記載の装置。

【請求項６８】　有意プロパゲーション・パスを実行する方法であって、
領域に対する有意状態情報に応じて、処理される情報が有意プロパゲーション・パス、リファインメント・パス、又は、クリーンアップ・パスに属するかどうかを判定する手順と、
領域内のサブ領域が、有意プロパゲーション・パス、リファインメント・パス、又は、クリーンアップ・パスの係数を有するかどうかを表わす信号をアサートする手順と、
リファインメント・パス及びクリーンアップ・パスに対するランカウント及びスキップカウントを識別すると共に、有意プロパゲーション・パスの係数のビットを符号化する手順と、
リファインメント・パスにあるものとして認定された係数のビット及びクリーンアップ・パスにあるものとして

認定された係数のビットを処理する手順と、を有する方法。

【請求項６９】　先行ビットが有意プロパゲーション・ビットであり、現在ビットが有意プロパゲーション・ビットではないとき、リファインメント・パス又はクリーンアップ・パスのいずれかに対する新しいランを開始する手順を更に有する請求項６８記載の方法。

【請求項７０】　ランカウント情報及びスキップカウント情報を保持するデータ構造内のテーブルへのインデックスを、新しいエントリーへ調節する手順と、
新しいエントリーを新しいランの開始を表わす状態へ初期化する手順と、を更に有する請求項６９記載の方法。

【請求項７１】　新しいエントリーを初期化する手順は、スキップ標識を零へ初期化し、ラン標識をある値へ初期化する、請求項７０記載の方法。

【請求項７２】　各サブ領域は４×４領域により構成される、請求項６８記載の方法。

【請求項７３】　各サブ領域で有意係数の個数をカウントする手順を更に有する請求項６８記載の方法。

【請求項７４】　カウントが各サブ領域のサイズと一致する場合、各サブ領域における全てのビットをリファインメントとして処理する手順を更に有する、請求項６８記載の方法。

【請求項７５】　カウントが零と一致し、有意な近傍が存在しない場合、各サブ領域における全てのビットをクリーンアップとして処理する手順を更に有する、請求項６８記載の方法。

【請求項７６】　有意プロパゲーション・パスを実行する装置であって、
領域に対する有意状態情報に応じて、処理される情報が有意プロパゲーション・パス、リファインメント・パス、又は、クリーンアップ・パスに属するかどうかを判定する手段と、
領域内のサブ領域が、有意プロパゲーション・パス、リファインメント・パス、又は、クリーンアップ・パスの係数を有するかどうかを表わす信号をアサートする手段と、
リファインメント・パス及びクリーンアップ・パスに対するランカウント及びスキップカウントを識別すると共に、有意プロパゲーション・パスの係数のビットを符号化する手段と、
リファインメント・パスにあるものとして認定された係数のビット及びクリーンアップ・パスにあるものとして認定された係数のビットを処理する手段と、を有する装置。

【請求項７７】　先行ビットが有意プロパゲーション・ビットであり、現在ビットが有意プロパゲーション・ビットではないとき、リファインメント・パス又はクリーンアップ・パスのいずれかに対する新しいランを開始する手段を更に有する請求項７６記載の装置。

【請求項７８】　ランカウント情報及びスキップカウント情報を保持するデータ構造内のテーブルへのインデックスを、新しいエントリーへ調節する手段と、
新しいエントリーを新しいランの開始を表わす状態へ初期化する手段と、を更に有する請求項７７記載の装置。

【請求項７９】　新しいエントリーを初期化する手段は、スキップ標識を零へ初期化し、ラン標識をある値へ初期化する、請求項７８記載の装置。

【請求項８０】　各サブ領域は４×４領域により構成される、請求項７６記載の装置。

【請求項８１】　各サブ領域で有意係数の個数をカウントする手段を更に有する請求項７６記載の装置。

【請求項８２】　カウントが各サブ領域のサイズと一致する場合、各サブ領域における全てのビットをリファインメントとして処理する手段を更に有する、請求項７６記載の装置。

【請求項８３】　カウントが零と一致し、有意な近傍が存在しない場合、各サブ領域における全てのビットをクリーンアップとして処理する手段を更に有する、請求項７６記載の装置。

【請求項８４】　有意プロパゲーション・パスを実行する装置であって、
第１の所定のサイズの第１の領域に対する有意状態情報を受け取る入力、並びに、第２の所定のサイズの第２の領域における各係数と関連したパスが有意プロパゲーション・パスであるか、リファインメント・パスであるか、又は、クリーンアップ・パスであるかを示す第１の出力標識の組を具備した決定パスユニットと、
出力標識に接続された入力、次の非リファインメント・ビット出力、次の非クリーンアップ・ビット出力、及び、現在のパス標識出力を具備した処理ユニットと、
処理ユニットからの出力へ接続され、上記処理ユニットからの出力に応じて、リファインメント・ビットデータ構造への次のインデックスの標識、リファインメント・ラン標識、リファインメント・スキップ標識、及び、有意プロパゲーション標識を生成する制御ユニットと、が設けられた装置。

【請求項８５】　上記処理ユニットは、
第１の出力標識の組、及び、処理されている第２の領域の現在の係数を表わすカウントを受け取り、現在の係数に対する第２の出力標識の組を具備し、第２の出力標識の組のうちの一つの出力標識だけが現在の係数に対してアサートされている、選択ユニットと、
決定パスユニットからのリファインメント・パス標識及びクリーンアップ・パス標識とカウントを受け取り、既に処理された第２の領域における係数と関連したリファインメント・パス標識及びクリーンアップ・パス標識のビットをマスクすることにより、マスク型リファインメント・パス標識及びマスク型クリーンアップ・パス標識を表現する出力の対を有するマスクユニットと、

マスク型リファインメント・パス標識及びマスク型クリーンアップ・パス標識を受ける入力を具備し、有意プロパゲーション・パスに対する次の非リファインメント・ビット位置及び次の非クリーンアップ・ビット位置を表現する出力の対を具備した優先度符号器と、を有する、請求項８４記載の装置。

【請求項８６】　制御ユニットは、処理ユニットからの出力に応答し、クリーンアップ・ビットデータ構造への次のインデックスの標識、クリーンアップ・ラン標識、及び、クリーンアップ・スキップ標識を生成する、請求項８４記載の装置。

【請求項８７】　現在のビットが有意プロパゲーション・パスに属する場合、係数は有意プロパゲーション・パスで処理される、請求項８４記載の装置。

【請求項８８】　現在のビットがリファインメント・パスに属する場合、リファインメント・パスのランレングスであるＫと一致するＫ個のビットがリファインメント・パスで処理される、請求項８４記載の装置。

【請求項８９】　現在のビットがクリーンアップ・パスに属する場合、クリーンアップ・パスのランレングスであるＫと一致するＫ個のビットがクリーンアップ・パスで処理される、請求項８６記載の装置。

【請求項９０】　次の非リファインメント・ビットが現在の係数よりも時間的に後で処理されるべきとき、制御ユニットは、ランレングスと一致するリファインメント・ラン標識を設定し、第２の領域での係数のスキップは、現在の係数に基づいて行なわれるべきではない旨を指示するようにリファインメント・スキップ標識を設定する、請求項８４記載の装置。

【請求項９１】　制御ユニットは、符号ブロック内でのリファインメント・ビットの場所を指示するランカウント及びスキップカウントを保持するデータ構造へのインデックスを増加させるべき旨を指示するようにリファインメントの次のインデックス標識を設定する、請求項９０記載の装置。

【請求項９２】　次の非リファインメント・ビットが現在の係数よりも時間的に後で処理されるべきとき、制御ユニットは、ランレングスと一致するリファインメント・ラン標識を設定し、第２の領域での係数のスキップは、現在の係数に基づいて行なわれるべきではない旨を指示するようにリファインメント・スキップ標識を設定し、

さらに、制御ユニットは、符号ブロック内でのリファインメント・ビットの場所を指示するランカウント及びスキップカウントを保持するデータ構造へのインデックスを増加させるべき旨を指示するようにリファインメントの次のインデックス標識を設定し、

さらに、制御ユニットは、クリーンアップ・ビットの現在のランが現れていない旨を指示するクリーンアップ・ラン標識を設定し、次の非リファインメント・ビット位置から第２の領域における現在係数の位置に等しい数を差し引いた値に一致するクリーンアップ・スキップ標識を設定する、請求項８６記載の装置。

【請求項９３】　制御ユニットは、符号ブロック内でのクリーンアップ・ビットの場所を指示するランカウント及びスキップカウントを保持するデータ構造へのインデックスを変更するべきではない旨を指示するようにクリーンアップの次のインデックス標識を設定する、請求項９２記載の装置。

【請求項９４】　有意プロパゲーション・インデックスは、現在の係数が有意プロパゲーション・パスに属さない旨を示すように設定される、請求項９３記載の装置。

【請求項９５】　次の非クリーンアップ・ビットが現在の係数よりも時間的に後で処理されるべきとき、制御ユニットは、ランレングスと一致するクリーンアップ・ラン標識を設定し、第２の領域での係数のスキップは、現在の係数に基づいて行なわれるべきではない旨を指示するようにクリーンアップ・スキップ標識を設定する、請求項８６記載の装置。

【請求項９６】　制御ユニットは、符号ブロック内でのクリーンアップ・ビットの場所を指示するランカウント及びスキップカウントを保持するデータ構造へのインデックスを増加させるべき旨を指示するようにクリーンアップの次のインデックス標識を設定する、請求項９５記載の装置。

【請求項９７】　制御ユニットは、さらに、クリーンアップ・ビットの現在のランが現れていない旨を指示するクリーンアップ・ラン標識を設定し、次の非リファインメント・ビット位置から第２の領域における現在係数の位置に等しい数を差し引いた値に一致するクリーンアップ・スキップ標識を設定する、請求項９６記載の装置。

【請求項９８】　制御ユニットは、符号ブロック内でのクリーンアップ・ビットの場所を指示するランカウント及びスキップカウントを保持するデータ構造へのインデックスを変化させるべきではない旨を指示するようにクリーンアップの次のインデックス標識を設定する、請求項９７記載の装置。

【請求項９９】　有意プロパゲーション・インデックスは、現在の係数が有意プロパゲーション・パスに属さない旨を示すように設定される、請求項９８記載の装置。

【請求項１００】　決定パスユニットは、ＮとＭが整数を表わすとき、処理される係数毎に１個ずつの所定の数のＮ×Ｍ領域を検査する、請求項８４記載の装置。

【請求項１０１】　Ｎ×Ｍ領域は３×３領域により構成される、請求項１００記載の装置。

【請求項１０２】　Ｎ×Ｍ領域の所定の数は１６個である、請求項１００記載の装置。

【請求項１０３】　第１の出力標識の組は、有意プロパゲーション・パス、リファインメント・パス、及び、クリーンアップ・パスの出力標識を含み、有意プロパゲー

ション・パス、リファインメント・パス、及び、クリーンアップ・パスの各出力標識は、１６ビット幅である、請求項８６記載の装置。

【請求項１０４】　有意プロパゲーション・パス、リファインメント・パス、及び、クリーンアップ・パスの各出力標識は、第２の領域の係数に対応した１ビットを有し、その中の唯一のビットだけが各サイクル中にアサートされる、請求項８６記載の装置。

【請求項１０５】　マスクユニットは、既に処理された係数に対応するリファインメント・パス及びクリーンアップ・パスの各標識の信号ラインをマスクする、請求項８６記載の装置。

【請求項１０６】　優先度符号器は零検出用優先度符号器を含む、請求項８５記載の装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、圧縮及び伸張の技術に係り、特に、ラン・スキップカウントに基づくコンテキストモデル・スキッピング技術及びメモリアクセス技術に関する。

【０００２】

【従来の技術】データ圧縮は、大量のデータを蓄積、伝送するための非常に有効なツールである。たとえば、文書のファクシミリ伝送のような画像を伝送するために要する時間は、画像を提示するために必要なビット数を減少させるため圧縮が使用されると、劇的に削減される。

【０００３】従来技術では、多種多様なデータ圧縮技術が知られている。圧縮技術は、損失のある符号化と損失の無い符号化の二つのカテゴリーに大別される。損失のある符号化には、原データの完全な再構成が保証されないような情報の損失を生じる符号化が含まれる。損失のある圧縮の目標は、原データへの変更が邪魔にならないか、又は、検出できないように行なわれることである。損失の無い圧縮の場合、全ての情報が保持され、完全な再構成ができるようにデータを圧縮することである。

【０００４】損失の無い圧縮の場合、入力シンボル又は強度データは、出力符号語に変換される。入力は、画像、オーディオ、１次元データ（たとえば、空間的に変化するデータ）、２次元データ（たとえば、空間的な２方向で変化するデータ）、又は、多次元／多重スペクトルデータを含む。圧縮が成功した場合、符号語は、入力信号（若しくは強度データ）の「ノーマル」表現におけるビット数よりも少ないビットで表現される。損失の無い符号化方法は、辞書的な符号化法（たとえば、Lempel-Ziv法）、ランレングス符号化法、エニュメラティブ(enumerative)符号化法、及び、エントロピー符号化法を含む。損失の無い画像圧縮の場合、圧縮は、予測又はコンテキストと符号化に基づく。ファクシミリ圧縮用のJBIG標準と、連続階調画像用のDPCM（差分パルス符号変調－JPEG標準における一つのオプション）は、損失の無い画像用圧縮の例である。損失のある圧縮の場合、入力シンボル又は強度データは、出力符号語へ変換する前に量子化される。量子化は、重要ではない特性を除去すると共に、関連性のあるデータの特性を保存するよう意図されている。量子化の前に、損失のある圧縮システムは、屡々、エネルギー集約を行なうため変換を使用する。JPEGは、画像データ用の損失のある符号化方法の一例である。

【０００５】可逆変換（ウェーブレット変換、コンポーネント変換）は、損失のある圧縮、及び、損失の無い圧縮の両方の圧縮に使用される。不可逆変換（ウェーブレット、コンポーネント、離散コサイン）は、損失のある圧縮だけに使用される。

【０００６】新たなJPEG 2000復号化標準は、変換を利用し、画像用の新しい符号化体系及び符号ストリーム規定を提供する。JPEG 2000標準は復号化標準であり、復号器のあるべき形を定義するが、この定義は、特に、損失の無い圧縮用の符号器を制約する。JPEG 2000標準の下で、各画像は矩形タイルに分割される。２個以上のタイルが存在する場合、画像のタイル張りによって、タイルコンポーネントが生成される。画像は多数のコンポーネントを含む。たとえば、カラー画像は、赤、緑及び青のコンポーネントを有する。タイルコンポーネントは、相互に独立に抽出若しくは復号化される。

【０００７】画像のタイル張り後、タイルコンポーネントは、ウェーブレット変換を用いて一つ以上の異なる分解レベルに分解される。これらの分解レベルは、原タイル・コンポーネントの水平及び空間周波数特性を記述する係数で占められた多数のサブバンドを収容する。係数は、画像全体ではなく、局所領域に関する周波数情報を与える。すなわち、少数の係数が単一のサンプルを完全に記述する。分解レベルは、サブバンドの連続した分解レベルの水平分解能及び垂直分解能が前の分解レベルの約半分になるように、空間的な倍率２で次の分解レベルに関連付けられる。

【０００８】サンプルと同数の係数が存在するが、情報内容は、数個の係数だけに集中する傾向がある。量子化によって、多数の係数の数値精度は、不釣合いに低い歪み（量子化ノイズ）の導入と共に減少する。エントロピー符号器による付加的な処理は、これらの量子化された係数を表現するため要求されるビット数を減少させ、場合によっては原画像よりも著しく減少させる。

【０００９】タイル・ンポーネントの個別のサブバンドは、さらに、符号ブロックに分割される。符号ブロックは、区域にグループ化される。これらの係数の矩形状アレイは、独立に抽出され得る。符号ブロック中の係数の個別のビットプレーンは、３段階の符号化パスでエントロピー符号化される。３段階のうちの各符号化パスは、ビットプレーン圧縮画像データに関するコンテキスト（前後関係）情報を収集する。

【００１０】これらの符号化パスから生成されたビットストリーム圧縮画像データは、レイヤにグループ分けされる。レイヤは、符号ブロックからの連続した符号化パスの任意のグループ分けである。レイヤ化は柔軟性に富んでいるが、連続した各レイヤがより高品質の画像に寄与することが前提である。各分解能レベルでのサブバンド係数の符号ブロックは、区域(precincts)と呼ばれる矩形領域に分割される。

【００１１】パケットは、圧縮符号ストリームの基本単位である。パケットは、一つのタイルコンポーネントの一つの分解能レベルの一つの予測レイヤからの圧縮画像データを含む。これらのパケットは、符号ストリーム中に定められた順序で配置される。

【００１２】タイルに関連した符号ストリームは、パケットに構成され、一つ以上のタイル部分に並べられる。タイル部分ヘッダは、マーカー及びマーカーセグメント、又は、タグの系列によって構成され、あらゆるタイルコンポーネントの位置を見つけ、抽出し、復号し、再構築するために必要な種々の仕組み及び符号化スタイルに関する情報を収容する。全符号ストリームの先頭には、マーカー及びマーカーセグメントにより構成された主ヘッダがあり、主ヘッダは、原画像に関する情報並びに類似した情報を提供する。

【００１３】符号ストリームは、アプリケーションが画像の意味、及び、画像に関するその他の情報の意味を解釈できるように、随意的にファイルフォーマットにまとめられる。このファイルフォーマットは、符号ストリーム以外のデータを収容する。

【００１４】JPEG 2000符号ストリームの復号化は、符号化手順の順序を逆転させることにより実行される。図１には、圧縮画像データ符号ストリームに作用するJPEG 2000標準符号化体系のブロック図が示されている。図１を参照するに、最初に、ビットストリームがデータ順序付けブロック１０１によって受信され、データ順序付けブロック１０１は、レイヤ及びサブバンド係数を再グループ分けする。算術符号器１０２は、前に符号化された、ビットモデリングブロック１０３から与えられたビット・プレーン圧縮画像データに関する係数及びビットモデリングブロック１０３の内部状態からのコンテキスト情報を使用して、圧縮ビットストリームを復号化する。

【００１５】次に、符号ストリームは、量子化ブロック１０４によって量子化され、量子化ブロック１０４は、ROI（領域選択画質向上）ブロック１０５によって示されるような領域選択画質向上処理(ROI)に基づいて量子化を行なう。量子化後、逆ウェーブレット／空間変換が変換ブロック１０６によって係数に適用され、その後に、DCレベルシフト及び選択的コンポーネント変換ブロック１０７が続く。これにより、再生画像が生成される。

【００１６】

【発明が解決しようとする課題】本発明は、情報を符号化及び復号化する方法並びに装置の提供を目的とする。

【００１７】

【課題を解決するための手段】一実施例において、本発明の装置は、メモリ及び復号化ハードウェアを含む。メモリは、ラン・カウント、及び／又は、スキップ・カウントを蓄積し、復号化ハードウェアはラン・カウント、及び／又は、復号化中にメモリから取得されたスキップ・カウントを復号化する。

【００１８】本発明は、以下の実施例の記載及び添付図面を参照することによって十分に理解されるであろう。しかし、実施例の記載及び添付図面は、本発明を特定の実施例に限定するものではなく、本発明の説明と解明を意図するものである。

【００１９】

【発明の実施の形態】符号化を実行する技術について説明する。符号化を実行する技術は、JPEG 2000標準を実現するため、特徴セットを操作するため、或いは、特徴セットへ追加するため使用される。すなわち、JPEG 2000標準として、参考のため引用したITU-T Rec. T.800|ISO/IEC FDIS 15444: 2000 JPEG Image Coding Systemに記載されている情報テクノロジー－JPEG 2000画像符号化システムであるCore Coding System（コア・コーディング・システム）は、導入者に多数の選択の範囲を残す。上記文献に記載された技術の目的は、ソフトウェア、ハードウェア及び／又はファームウェアにおいて、高速、低価格、小メモリ容量、及び／又は、性能豊富な手段を実現するため、JPEG 2000での選択の範囲を利用することである。

【００２０】以下では、本発明が完全に理解できるように、多数の細部について説明する。しかし、本発明がこれらの特定の細部を用いることなく実施されることは、当業者には明らかであろう。その他の場合、周知の構成及び装置は、本発明の本質がわかり難くなることを避けるために、詳細図ではなくブロック図の形式で示されている。さらに、詳細には記載されていないブロック、ロジック又は機能は、周知の手段を用いて実現されるか、或いは、当業者によって周知のハードウェア、ソフトウェア及び／又はファームウェアを用いて容易に実現される。一部の技術及び手段は、疑似コードを用いて記述されていることに注意する必要がある。しかし、このことは、この一部の技術及び手段が、ソフトウェアだけによって実現されることを意味するわけではなく、むしろ、このような記載は、当業者が容易に理解するであろう項目の機能を簡単に説明するために、屡々、選択される。

【００２１】以下の詳細な説明の一部分は、コンピュータメモリ内のデータビットに対する演算のアルゴリズムと記号的表現とを用いて提示される。これらのアルゴリズム的記述及び表現は、データ処理技術の分野の当業者

が、自分の業績をその分野の当業者に最も効率的に伝えるために使用する手段である。ここで、アルゴリズムは、一般的に、所望の結果を導く一貫した手順のシーケンスであると考えられる。これらの手順は、物理量の実際的な操作を必要とする手順である。通常、必ずしも不可欠ではないが、これらの物理量は、記憶、転送、合成、比較、或いは、操作することが可能な電気信号又は磁気信号の形式で得られる。主として、一般的な用法であるとの理由から、これらの信号は、ビット、値、要素、シンボル、文字、項、数などによって指定することが時に好都合であることがわかる。

【００２２】しかし、これらの用語及び類義語は、適切な物理量と関連付けられ、物理量に与えられた便宜的なラベルに過ぎない。特に断らない限り、以下の説明から明らかであるように、「処理」、「コンピューティング」、「計算」、「決定」、或いは、「表示」のような用語を利用する記述は、コンピュータシステム若しくは類似した電子コンピューティング装置の動作及び処理を示すものであり、コンピュータシステムのレジスタ及びメモリ内で物理（電子）量として表現されたデータを操作し、同じように、コンピュータシステムのメモリ若しくはレジスタ、又は、他の情報記憶装置、情報伝送装置、若しくは、情報表示装置内で物理量として表現された他のデータへ変換する。

【００２３】本発明は、以下で説明する動作を実行する装置に関する。この装置は、特に、要求された用途に応じて構成されるか、或いは、選択的に作動され、若しくは、コンピュータに記憶されたコンピュータプログラムを用いて再構成される汎用コンピュータを含む。このようなコンピュータプログラムは、たとえば、フレキシブルディスク、光ディスク、ＣＤ－ＲＯＭ、光磁気ディスクなどを含む任意のタイプのディスクや、読み出し専用メモリ（ＲＯＭ）や、ランダム・アクセス・メモリ（ＲＡＭ）や、ＥＰＲＯＭや、ＥＥＰＲＯＭや、磁気若しくは光カードや、電子命令を記憶するため適した任意のタイプの媒体のような、コンピュータシステムバスに接続された、コンピュータ読み取り可能な記憶媒体に記憶されるが、これらの例示的な記憶媒体に制限されるものではない。

【００２４】ここで説明するアルゴリズム及びディスプレイは、本来的に特定のコンピュータ若しくはその他の装置に関連付けられたものではない。種々の汎用システムが、ここで教示された事項に応じたプログラムと共に使用される。或いは、要求された方法の手順を実行するため、より専用化された装置を構成した方が便利な場合もある。多様なこれらのシステムに対し要求される構成は、以下の記載から明らかになる。さらに、本発明は、特定のプログラミング言語に基づいて説明されていない。以下で説明するような本発明の教示事項を実現するために、多様なプログラミング言語を使用できることが認められるであろう。

【００２５】機械読み取り可能な媒体は、機械（たとえば、コンピュータ）によって読み取り可能な形式で情報を記憶若しくは伝送する任意のメカニズムを含む。たとえば、機械読み取り可能な媒体は、読み出し専用メモリ（ＲＯＭ）と、ランダム・アクセス・メモリ（ＲＡＭ）と、磁気ディスク記憶媒体と、光記憶媒体と、フラッシュメモリ装置と、電気的、光学的、音響的若しくはその他の形式の伝搬信号（たとえば、搬送波、赤外線信号、デジタル信号など）と、を含む。

【００２６】〔概要〕図２９は、符号器の一実施例のブロック図である。図２９を参照するに、データ・インタフェース（Ｉ／Ｆ）２９０１は、符号化されるべきデータ、或いは、復号化後に出力されるべきデータを受信するため接続される。ＤＣレベルシフタ２９０２は、データ・インタフェース２９０１に接続され、符号化中及び復号化中にＤＣレベルシフトを実行する。ウェーブレット変換器２９０３は、ＤＣレベルシフタ２９０２に接続され、処理フローの方向に依存して、順ウェーブレット変換又は逆ウェーブレット変換を実行する。一実施例において、ウェーブレット変換器２９０３は、５，３－可逆ウェーブレット変換、及び、５，３－不可逆ウェーブレット変換を実行し、画像を２乃至５レベルに分解する。ラインバッファ２９０４は、ウェーブレット変換器２９０３に接続され、ウェーブレット変換を実行する際に、ウェーブレット変換器２９０３にデータを供給する。

【００２７】スカラー量子化／逆量子化ブロック２９０５は、ウェーブレット変換器２９０３に接続され、スカラー量子化を実行する。一実施例において、スカラー量子化は、５，３－ウェーブレット変換だけのため使用される。プリコーダ２９０６は、スカラー量子化ブロック２９０５に接続され、プリコーディングを行なう。一実施例において、プリコーダ２９０６は、係数を２の補数から符号付きの尺度（大きさ）に変換する（復号化の場合には、逆変換する）。プリコーダは、零ビットプレーンを決定する。ワークメモリＡ及びワークメモリＢは、パケットヘッダ処理器２９０７と共に、プリコーダ２９０６に接続される。ワークメモリＡ及びワークメモリＢと、パケットヘッダ処理器２９０７へのインタフェースは、ビットモデリングＭＱ符号器2908_{1-N}にも接続される。各ＭＱ符号器2908_{1-N}は、符号化データ(JPEG 2000の用語では圧縮データ）を保持する個別の符号メモリ2911_Nに接続される。符号メモリからの符号化データと、パケットヘッダ処理器２９０７からのパケットヘッダは、符号化データとして出力される。これは、JPEG 2000ビットストリームである。付加的な機能ブロック（図示しない）は、主ヘッダ及びタイル部ヘッダの作成／読出しのため使用される。ビットストリーム及びヘッダは、JPEG 2000符号ストリームを形成す

る。

【００２８】〔サブビットプレーンに対するスキップを含むコンテキスト・モデル・データ構造〕JPEG 2000において、（最上位ビット（ＭＳＢ）から始めて下位ビットへ）初期的に全零ではない係数のビットプレーンに対して、各係数は、有意プロパゲーション、リファインメント、クリーンアップの三つのサブビットプレーンパスのうちの一つのパスで符号化される。図２Ａには、係数毎に、１ビットプレーンのサブビットプレーンパスが識別された８×８符号ブロックの一例が示されている。図２Ａを参照すると、ＳＰは、有意プロパゲーション・パスを表わし、Ｒは、リファインメント・パスを表わし、Ｃは、クリーンアップ・パスを表わす。図２Ａにおける指数０～６３は、符号ブロック走査順序を示す。このようにして、走査順序は、下向きに４係数、次に、最上位行に戻るという動作を符号ブロック全体に亘って繰り返す。符号ブロック全体の走査が終了した後、次に、走査は、各列の５番目の係数から下向きに、残りの符号ブロックの全域に亘って続けられる。

【００２９】典型的な実現例では、ビットプレーンの符号化パス毎に１回ずつの合計３回に亘って係数のブロック全体を読み出す。この技術は、各ビットプレーンの有意プロパゲーション・パスに対し係数ブロックの全体を読み出し、リファインメント・パスとクリーンアップ・パスに対して実際に必要とされる係数だけを読み出す方法を説明する。

【００３０】図２Ａの各セルの左側において、実線は、リファインメント・サブビットプレーンパスの係数を示し、点線は、リファインメント・サブビットプレーンパスでスキップされた係数を表わす。図２Ａの各セルの右側の実線は、クリーンアップ・サブビットプレーンパスに対する係数を示す。パスが係数毎に識別されると、係数が処理される。

【００３１】データ構造は、後述の処理を用いて、有意プロパゲーション・パス中に構築される。このデータ構造は、メモリへのアクセス回数を低減させるため、コンテキストモデルによって使用される。このデータ構造を用いることによって、情報が属するパスがどのパスであるかを判定するためセル毎に検査し、セルをスキップする必要があったのに対して、コンテキストモデルは、メモリに１回だけアクセスすればよい。さらに、このデータ構造は、たとえば、クリーンアップ・ビットが同時に＊４ビットだけ符号化されるときのように、多数の場所へ同時にアクセスすることを可能にさせる。

【００３２】表１及び表２は、それぞれ、リファインメント・サブビットプレーンパスにおける係数の場所及びクリーンアップ・サブビットプレーンパスにおける係数の場所を記述する。インデックス毎に、サブビットプレーンパスにおける係数の個数のランカウントと、異なるパスに含まれる後続係数の個数のスキップカウントが示されている。これらのデータ構造は、サブビットプレーンパスを効率的に符号化することを可能にさせる。そのために、他のパスにおける係数をスキップできるようにする。

【００３３】

【表１】

表１
リファインメントビット用データ構造

インデックス	ラン	スキップ
0	0	1
1	2	8
2	1	3
3	1	48

【００３４】

【表２】

表２
クリーンアップビット用データ構造

インデックス	ラン	スキップ
0	0	8
1	2	2
2	2	2
3	2	2
4	16	1
5	3	1
6	3	1
7	3	1
8	15	0

この処理は、ハードウェア、ソフトウェア、又は、両方の組み合わせを含む処理ロジックによって実行される。一実施例において、データ構造を作成し、データ構造を使用するこの処理ロジックは、図２９のビットモデルングＭＱ符号器２９０８$_{1-N}$　に設けられる。処理中に、これらのデータ構造を作成するため、データ構造が最初に初期化される。

【００３５】

```
ri=0                    //リファインメント用インデックス
ci=0                    //クリーンアップ用インデックス
r_run[ri]=0             //リファインメント用ランカウント
r_skip[ri]=0            //リファインメント用スキップカウント
c_run[ci]=0             //クリーンアップ用ランカウント
c_skip[ci]=0            //クリーンアップ用スキップカウント
state=INITIAL           //状態stateは、INITIAL、SIG_PROP、REFINE又はCLEAN
UPである。
```

【００３６】状態変数は、実行（ラン）の開始と中央を区別するため使用される。状態変数は、前の係数用の符号化パスを示す。現在の係数が同じである場合、ラン若しくはスキップのサイズは増加し、異なる場合、新しい＊ランが開始される。符号ブロックの各係数は、別々のカウントを生成するため、符号ブロック走査順に考慮される。

【００３７】

```
For y1=0 to maximum-for-y1 step 4
    for x=0 to maximum-for-x step 1
        for y2=0 to maximum-for-y2 step 1
            process coefficient[x,y1+y2]
```

上記コードにおいて、y1の最大値は、符号ブロックの高さ（"(height-1)&〜3"）よりも小さい４の最大の整数倍である。xの最大値は、符号ブロックの"width-1"である。y2の最大値は、３と"height-y1-1"の小さい方である＊※る。各係数を処理する手続の一実施例は、以下の通りである。

【００３８】

```
if  係数が前のビットプレーンにおいて有意である、then
    if  stateがREFINEでない、then
        ri=ri+1
        r_run[ri]=1
        r_skip[ri]=0
        state=REFINE
    else
        r_run[ri]=r_run[ri]+1
    c_skip[ci]=c_skip[ci]+1
else  if  係数の近傍が有意である、then
    (係数は予測された有意な符号)
    r_skip[ri]=r_skip[ri]+1
    c_skip[ci]=c_skip[ci]+1
    state=SIG_PROP
else
    stateがCLENUPではない、then
        ci=i+1
        c_run{ci]= 1
        c_skip[ci]=0
        state=CLEANUP
    else
        c_run[ci]=c_run[ci]+1
    r_skip[ri]=r_skip[ri]+1          //。
```

【００３９】この手続を適用した結果、有意プロパゲーション・パスの全係数は符号化され、リファインメント・ビット及びクリーンアップ・ビットに対するデータ構造が作成される。

【００４０】必要に応じて、ランカウントは、行を超えて元に戻らないようにすることができる。この行のラップアラウンドを防止するための処理の一実施例は、以下の疑似コードに記載される。これにより、境界を非常に簡単に取り扱うことが可能になる。

【００４１】

```
for y1=0 to maximum-for-y1 step 4
    for x=0 to maximum-for-x step 1
        for y2=0 to maximum-for-y2 step 1
            process coefficient[x,y1+y2]
    if  stateがREFINEである、then
        ri=ri+1
        r_run[ri]=0
        r_skip[ri]=0
        state=INITIAL
    else  if  stateがCLEANUPである、then
        ci=ci+1
        c_run[ci]=0
        c_skip[ci]=0
        state=INITIAL                //。
```

【００４２】ソフトウェアの場合、ラン値及びスキップ

値を整数（３２ビットコンピュータの場合には、３２ビットなど）として保持することが最も好都合である。最悪のケースは、長さ０のランでスタートした長さ１のランである。JPEG 2000において、符号ブロックの係数は、最大で４０９６個に制限される。符号ブロックの幅と高さも最大で１０２４個の係数に制限される。全部で４０９６個の係数の任意のサイズの符号ブロックに対し、ランカウントが行のグループの全域で継続する場合、４０９７個のメモリロケーションが、そのメモリサイズに対するメモリロケーションの最大数である。６４×６４の符号ブロックに対し、ランカウントが４行ずつのグループ毎にスタートする場合、（４×６４＋１）×（６４／４）＝４１１２個のメモリロケーションが最大値である。１０２４×４の符号ブロックに対し、ランカウントが４行ずつのグループ毎にスタートする場合、（４×４＋１）×（１０２４／４）＝４３５２個のメモリロケーションが最大値である。

【００４３】ハードウェアのメモリを節約するため、最小限の固定数のビットがランカウント及びスキップカウントのため使用され得る。最初のカウントがランカウントとスキップカウントのどちらであるかを示す標識が信号（たとえば、１ビット信号標識）で示されたとき、ランカウントは１よりも大きい（かつ、０を符号化する能力は要求されない）。全部で４０９６個の係数をもつ任意のサイズの符号ブロックに対して、ライン（行）のグループに対してランカウントが継続する場合、最初のカウントがランであるか、又は、スキップであるかを示すために、１ビットが使用され、４９，１５３ビット全体に対して、４０９６×１２ビットが使用される。６４×６４符号ブロックに対して、ランカウントが４行ずつのグループ毎に始まる場合、４行ずつの各グループに対し最初のカウントがランであるかスキップであるかを示すため１ビットが使用される。かくして、ビット数は、１×６４／４＋４０９６×１２＝４９，１６８ビットである。１０２４×３の符号ブロックに対し、ランカウントが４行のグループ毎に始まる場合、ビット数は、１×１０２４／４＋４０９６×１２＝４９，４０８ビットである。

【００４４】可変長符号の一実施例は、カウントを表現するため使用され得る。表３では、小さいカウントが少数のビット（たとえば、３ビット）で表現され、大きいカウントは、多数のビット（たとえば、１３ビット）で表現される。このようなアプローチの目標は、比較的小さい符号語を頻繁に使用できるように、殆どのカウントを、１、２、３若しくは４で表わすことである。より簡単に実現するために、二つのサイズだけが使用される。しかし、三つ以上のサイズを使用して、複雑さを増しても構わない。

【００４５】

【表３】

表３

簡単なカウント用の可変長符号の一例

カウント	符号語
1	000
2	001
3	010
4	011
5	1000000000000
6	1000000000001
...	...
4096	1111111111011

この符号の場合、全てのランレングスが１（すなわち、全符号語が３ビット）となるときに最悪の状況が生ずる。３通りの状況（行を横断してカウントする状況、４行ずつのグループによる６４×６４符号ブロックの状況、４行ずつのグループによる１０２４×４符号ブロックの状況）に対し、総ビット数は、それぞれ、１２，２８９ビット、１２，３０４ビット、及び、１２，５４４ビットになる。

【００４６】メモリ使用量の削減は、もっと複雑な可変長符号を用いることによって達成される。優れた構造の符号は、表４に示されるような、ガンマ符号、γ' 又は γ（BBell, Cleary, Whitten "Text Compression", Prenice Hall, NJ, 1990のAppendix A）である。

【００４７】

【表４】

表４　カウント用の構造化可変長符号

カウント	符号語（γ'フォーマット）	符号語（γフォーマット）
1	0	0
2	10_0	100
3	10_1	110
4	110_00	10100
5	110_01	10110
6	110_10	11100
7	110_11	11110
8	1110_000	1010100
9	1110_001	1010110
...	...	...
15	1110_111	1111110
16	11110_0000	101010100
32	111110_00000	10101010100
64	1111110_000000	1010101010100
128	11111110_0000000	101010101010100
256	111111110_00000000	10101010101010100
512	1111111110_000000000	101010101010101010100
1024	11111111110_000000000	1010101010101010101000
2048	111111111110_0000000000	10101010101010101010100
4096	1111111111110_00000000000	101010101010101010101000

γ' とγは、符号語のビットの並べ方だけが相違する。γ' 符号語中の"_"は、符号語の一部ではなく、カウンタからプレフィックスを分離することによって見易くさせるためのものである。最悪の状況は、３ビットを必要とする２のカウントの場合に生じる。３通りの状況（行を横断してカウントする状況、４行ずつのグループによる６４×６４符号ブロックの状況、４行ずつのグループによる１０２４×４符号ブロックの状況）に対し、必要とされる総ビット数は、それぞれ、６，１４５ビット、６，１６０ビット、及び、６，４００ビットになる。

【００４８】表５は、最長符号語が１６ビットで表わされるカウント１．．．４０９６に対する符号の一実施例である。カウント１、２、３、４又は５は、それぞれ、１、２、３、４及び５によって表現される。カウント６～２１は、８ビットで表わされる。カウント２２～４０９６は、１６ビットで表わされる。３及び６のカウントは、最悪の状況である。３通りの状況（行を横断してカウントする状況、４行ずつのグループによる６４×６４符号ブロックの状況、４行ずつのグループによる１０２４×４符号ブロックの状況）に対し、総ビット数は、それぞれ、５，４６３ビット、５，４７８ビット、及び、５，７１８ビットである。

【００４９】

【表５】

表５　カウント用の最適可変長符号

カウント	符号語
1	0
2	01
3	1100
4	1101_0
5	1101_1
6	1110_0000
7	1110_0001
...	...
21	1110_1111
22	1111_0000_0000_0000
23	1111_0000_0000_0001
...	...
4096	1111_1111_1110_1010

ハードウェアによる可変長符号を使用する場合、同じクロックサイクル中にランカウントとスキップカウントの両方へアクセスすることが望ましい。一実施例において、メモリは、最小サイズよりも１語（たとえば、１６ビット）分だけ大きいので、フラッシュが簡単である。たとえば、１６ビット語の場合、ランカウントは１６３ビットを使用する。したがって、最後の語は３ビットだけを使用する。これにより、１語を完成するために、１３ビットをパディングする必要が生じる。スキップカウントは８５ビットを使用する。したがって、最後の語は、５ビットしか使用しない。このため、１語を完成するには、１１ビットをパディングする必要がある。メモリサイズが、

メモリサイズ≧（（１６３＋１３）／１６）＋（（８５＋１１）／１６）

すなわち、

メモリサイズ≧１７

であるならば、ランカウントとスキップカウントのパディングは独立でもよい。サイズが１６であるならば、１語は、ランカウントと、スキップカウントと、両者の間のパティングとを含む。図３０には、ランカウントとス

キップカウントの両方を備えた１６ビット語の一例が示されている。

【００５０】図２Ｂは、可変長ラン・スキップカウント用のメモリの説明図である。このメモリ構造によれば、ランカウントをメモリの一方側（２０１）から開始し、スキップカウントをメモリの他方側（２０２）から開始させることが可能である。メモリの一方側２０１は、始まりでも終わりでもよく、他方側２０２は、終わりでも始まりでもよい。これにより、１ラン可変語と１スキップ可変語のスタートは、同時にわかっているので、ランカウントとスキップカウントを同時並行的に復号化できるようになる。そのため、ランカウントを先に復号化し、その長さを判定し、次に、スキップカウントを復号化しなくてもよい。直列的である場合（ラン・スキップ・ラン、以下同様である場合）、（一方を復号化しない限り他方を見つけられないので）一つずつ判明することになる。

【００５１】別個のスキップカウント及びランカウントは、図２Ｂに示されたメモリで情報を復号化するため使用される。或いは、１台の復号器をスキップカウントとランカウントの両方で共用してもよい。

【００５２】〔コンテキストモデル用ハードウェア〕コンテキストモデルはハードウェアに実現される。ハードウェアにおいて、一つの目標は、ＭＱ符号器が待機状態にならないように、できる限り早く次のコンテキストを生成することである。

【００５３】〔メモリ構成〕図３の（Ａ）～（Ｄ）は、コンテキストモデルの実施例のための近傍係数及びメモリ構成を示す図である。個別の係数用のコンテキストモデルは、図３Ａに示されるように、高々、３×３近傍に基づいている。一実施例において、４ビットが同時に処理される。このような状況において、４個の係数３１１～３１４のグループに対するコンテキストモデルは、図３Ｂに示されるように、高々、３×６近傍に基づいている。ハードウェアへのメモリアクセスは、２のべき乗でグループ化されたデータを処理することが屡々望ましい。したがって、３×６領域を含むような２のべき乗に基づく領域は、４×８領域である。図３の（Ｃ）は、係数の４×８領域の部分集合である３×６近傍を示す図である。図３の（Ｃ）の４×８領域全体へのアクセスは、＊同一のメモリ若しくは同一ではないメモリはの別々のアクセスとして実行される。図３の（Ｄ）は、４個の２×４領域３０１～３０４に分割された４×８領域を示す。２×４領域は、並行したランダムアクセスのため、異なるメモリへ個別に格納される。このメモリ構造によって、係数のグループからのコンテンツを判定するため必要な全ての情報をメモリから、順番にではなく、同時に読み出すことが可能になる。すなわち、係数情報の４×８ブロック全体は、同時にアクセスされる。

【００５４】図４は、１６×１６符号ブロックに対するランダムアクセス用の有意メモリ構成の一実施例を示す図である。一実施例では、たとえば、３２×３２、６４×６４のような他のサイズを取り扱うことができるが、これらのサイズには限定されない。図４を参照するに、各係数は、４個のメモリ（Ａ、Ｂ、Ｃ又はＤ）のうちの一つに割り当てられる。一部のグループ（一番上の２行と一番下の２行）は、他のグループのサイズの半分のサイズである。その理由は、図３の（Ｄ）において、符号ブロックの最初の２行に対し、一番上の２行が符号ブロックの外側にある（エッジから外れている）からである。同様な境界条件は、符号ブロックの一番下にも現れる。

【００５５】一実施例において、メモリは、有意状態用の係数に対し１ビット（１アドレスについて全部で８ビット）を記憶する。他の一実施例において、メモリは、有意状態用の係数とサイン毎に２ビット（１アドレスについて全部で１６ビット）を記憶する。さらに別の一実施例において、これらのメモリは、全係数（Ｎが１個の係数のサイズを表わすとき、８Ｎビット）を記憶する。他の一実施例において、全部の係数がこれらのメモリに記憶されない場合、１係数毎に１個のアドレスを備えた補助的な一つのメモリが使用される。

【００５６】以下のベリログ(Verilog)コードは、２個の６ビットアドレス入力（"x"及び"y"）を、（６４×６４符号ブロック用）コンテキストモデルの制御ロジックから、メモリＡ、Ｂ、Ｃ、若しくはＤを指定するための、メモリへの７ビットアドレス出力("addr")と、２ビットバンク選択とに変換する、一実施例である。

【００５７】

```
module makeAddress(x,y,addr,bank);
    input [5:0] x;          /*  xは、ビット５～０をもち、ビット５がMSBである */
    input[5:0] y;
    output[6:0] addr;
    output[1:0] bank;
    wire[5:0] yp2;
    assign yp2 = y + 2;
    assign addr = {yp2[5:3],x[5:2]}
    assign bank = {yp2[2],x[1]}
```

endmodule

【００５８】１番目のアサイン文は、境界に対するオフセットを設定する。すなわち、オフセット"assign yp2 = y + 2"は、図４に示されるように４行のグループの適切な配置を行なうため使用される。２番目のアサイン文は、入力yのビット５～３と、入力xのビット５～２と連結したオフセットの合計へのアドレスを、数の下位部分としてセットする。３番目のアサイン文は、入力yのビット２と、入力xのビット１と連結したオフセットの合計と一致するバンクを設定する。

【００５９】〔有意プロパゲーション・パス〕図５は、ランダムアクセス用の有意プロパゲーション・パスに使用されるメモリ及びレジスタの一実施例を示す図である。図５によれば、アドレスＡは、データ出力を生成するためメモリＡへ入力され、データ出力はレジスタ５０１にも保持される。アドレスＢに応答して、メモリＢは、データを出力し、出力されたデータは、レジスタレジスタ５０２にも保持される。同様に、メモリＣは、アドレスＣに応答してデータを出力し、出力はレジスタ５０３に保持される。最後に、メモリＤは、アドレスＤに応答してデータを出力し、そのデータはレジスタ５０４に記憶される。一実施例において、各メモリＡ～Ｄの出力は、２×４領域であり、全体として、４×８領域（たとえば、図６の領域６０１）を作成する。

【００６０】図５に示されたメモリ及びレジスタの全出力は、全体として、有意ビットの６×６領域を与える。＊

```
address_A_y=0
address_B_y=0
address_C_y=0
address_D_y=0
for y=0 to 60 step 4
    address_A_x=0
    address_C_x=0
    read memory A （次に登録される）
    read memory C （次に登録される）
    assert clear for memory B register （次にクリアされる）
    assert clear for memory D register （次にクリアされる）
    for x=0 to 60 step 4
        address_A_x=x+4
        address_B_x＝x
        address_C_x＝x
        address_D_x=x
        if x ＜ 60 then
            read memory A （次に登録される）
            read memory C （次に登録される）
        else
            use "all bits zero" for memory A output
            use "all bits zero" for memory B output
        read memory B （次に登録される）
        read memory D （次に登録される）
```

```
/* おわり。
```

＊これは、他の実施例では、有意状態及びサイン、又は、実際の係数になり得ることに注意する必要がある。すなわち、メモリＡ～Ｄからの並列に使用されるデータは、前のサイクルにメモリＡ～Ｄから読み出されレジスタ５０１～５０４に保持されたデータと組み合わされる。この有意ビットと、コンテキストモデルからのフィードバックを合わせた領域は、係数の４×４領域が存在するパスを決定するために十分な領域である。

【００６１】図６は、メモリから読み出され、ランダムアクセスのためレジスタに保持された有意状態を示す図である。図６によれば、領域６０１は、メモリから読み出された４×８領域を示す。領域６０２は、メモリＡ～Ｄから読み出され、コンテキストモデリングのため使用される３×６領域を示す。領域６０３は、レジスタ５０１から５０４に記憶され、コンテキストモデリングに使用される３×６領域を示す。領域６０４は、処理される係数の４×４領域を示す。図６には、メモリＡ～Ｄのメモリロケーション及びレジスタから得られた８×８ブロックの典型的な２×４部分が示されている。

【００６２】有意プロパゲーション・パス用のアドレス生成ロジックの一実施例は、以下の疑似コードに記載される。アドレッシングはデータに依存しないこと、及び、零データは境界に設けられることに注意する必要がある

```
    係数x...x+3,y...y+3の4x4ブロックを処理
if y AND 4==0
    address_A_y=address_A_y+8
    address_B_y=address_B_y+8
else
    address_C_y=address_C_y+8
    address_D_y=address_D_y+8          /*  。
```

【００６３】４×４ブロックを処理するため、同じパスのビットのランは一緒に取り扱われる。リファインメント・パスの行にＮ個の係数が存在する場合、以下の疑似コードがそれらを処理するため使用される。

【００６４】

```
if state is not REFINE then
    ri=ri+1
    r_run[ri]=N
    r_skip[ri]=0
    state=REFINE
else
    r_run[ri]=r_run[ri]+N          /*  。
```

【００６５】この疑似コードは、１個の係数ではなく、Ｎ個の係数が処理されていることを示すため、１の代わりにＮが使用されている点を除くと、上述のコードと類似していることに注意する必要がある。

【００６６】Ｎ個の係数がクリーンアップ・パスの行に存在する場合、以下の疑似コードは、係数を処理するためのプロセスの一実施例を表わす。

【００６７】

```
if state is not CLEANUP then
    ci=ci+1
    c_run[ci]=N
    c_skip[ci]=0
    state=CLEANUP
else
    c_run[ci]=c_run[ci]+N          /*  。
```

【００６８】図７は、有意プロパゲーション・パスロジックの一実施例のブロック図である。一実施例において、このロジックは、図２９のビットモデリングＭＱ符号器２９０８$_{1-N}$　に含まれる。係数毎のパスは、１係数当たりにつき、有意プロパゲーション又はそれ以外、リファインメント又はそれ以外、及び、クリーンアップ又はそれ以外の３ビットで表現された４×４領域に対するパスである。図５において、メモリＡ～Ｄのアクセス動作を制御することによって、４×４ブロックがメモリから取り出され、有意プロパゲーション・パスが実行される。各４×４ブロックを調べると、多数のパス中のランが識別され、有意パス中の係数を符号化する必要があり、一方、リファインメント・オア及びクリーンアップ・パスに対するランカウント及びスキップカウントは、同時に一つのランを処理するため識別される。ブロック内で（走査順による）前のビット（或いは、新しい符号ブロックの始まりの場合には前の符号ブロック）は、有意プロパゲーション・ビットであり、現在状態は有意プロパゲーションではなく、新しいランが始まる。このような場合に、インデックスは、ラン・スキップカウントを保持するテーブル内でインクリメントされる（たとえば、スキップを零にセットし、ランを第１の値にセットする）。ランカウントとスキップカウントの両方のテーブルは、このようにしてインクリメントされ、同時に、４×４ブロックを処理する。４×４ブロックの前のビットがリファインメント・パス又はクリーンアップ・パスにあり、より多くのこのようなデータが後に続くとき、現在ランのカウントはインクリメントされる。Ｎが２のべき乗を表わすとき、たとえば、４×Ｎ領域を含む他のサイズの領域を使用してもよい。

【００６９】図７を参照するに、８×８領域用の有意状態７０１は、パスを決定するロジック７０２への入力である。図３１は、典型的な８×８領域の説明図である。有意状態７０１は、たとえば、リファインメント・パスの行にＮ個の係数が存在することを示す情報を含む。このような情報は、上述のようなテーブルからアクセスされる。決定パスロジック７０２は、８×８領域の中央の６×６領域内の１６個の３×３領域を調べる。係数Ａ～Ｉは、１番目の３×３領域を表現する。図３２は、決定パスロジック７０２の一実施例の構成図である。図３２におけるロジックは、４×４ブロック内の各係数に対して１回ずつとして、１６回繰り返される。領域は、３×３領域以外のサイズでもよく、処理される領域の数は、同時に１６個より多くても少なくても構わないことに注意する必要がある。

【００７０】図３２を参照するに、係数Ａ～Ｃの全ビットは、ＯＲゲート３２０１へ入力される。係数Ｄ及びＦの全ビットは、ＯＲゲート３２０２の入力へ供給される。係数Ｇ～Ｉの全ビットは、ＯＲゲート３２０３の入力へ供給される。ＯＲゲート３２０１～３２０３の出力は、ＯＲゲート３２０４の入力へ接続される。ＯＲゲート３２０４の出力は、インバータ３２０６の入力と、ＡＮＤゲート３２０８の入力とに接続される。係数Ｅは、リファインメント信号７０４の１６ビット出力を表わし、インバータ３２０５の入力へ供給され、インバータ３２０５の出力はＡＮＤゲート３２０８の別の入力と、ＡＮＤゲート３２０７の入力とに供給される。ＡＮＤゲート３２０８の出力は、有意プロパゲーション信号７０３である。インバータ３２０６の出力は、ＡＮＤゲート

３２０７の他の入力へ供給される。ＡＮＤゲート３２０７の出力は、クリーンアップ信号７０５である。

【００７１】動作中に、有意状態ビットＥのいずれかが０である場合、ＡＮＤゲート３２０８の出力は、０であるビットのビット位置に対応し、有意状態が係数Ａ～Ｄ若しくは係数Ｆ～Ｉのいずれかに対して１である場合に、有意プロパゲーション信号７０４は、１に変化する。同様に、有意状態ビットＥのいずれかのビットが０である場合、ＡＮＤゲート３２０７の出力は、そのビットのビット位置に対応し、これにより、クリーンアップ信号７０５は、係数Ａ～Ｄ若しくは係数Ｆ～Ｉに対する有意状態ビットが全て０である場合、１に変化する。

【００７２】決定の結果として、ロジック７０２は、有意プロパゲーション信号７０３、リファインメント・パス信号７０４、又は、クリーンアップ・パス信号７０５のうちのいずれかをアサート（真の状態に）する。一実施例において、各信号７０３～７０５は、１６ビット幅である。信号７０３、７０４及び７０５における対応した各ビットに対し、唯一のビットが１であり、他の２個のビットが０である。かくして、１６個のロケーションに３通りの確率が存在する。ロジック７０２の各出力は、選択ロジック（たとえば、マルチプレクサ（ＭＵＸ））７０７の一方の入力へ供給される。

【００７３】選択ロジック７０７は、現在の係数に関して、現在の係数に対するパスを示す３個のパスビットを生成し、パスビットを制御ロジック７０９へ送る。３個のパスビットのうちの１ビットだけが、制御ロジック７０９から出力されたカウント信号７０８に応答してアサートされる。カウント信号７０８は、４×４ブロック内の１６個の係数のうちの何れの係数が現在処理中の係数であるかを示す。リファインメント・ビットラン及びクリーンアップ・ビットランを取り扱うとき、カウント信号７０８は１よりも大きい数でインクリメントされる。かくして、決定パスロジック７０２の各出力の１６ビットのうちで、その係数に対応した３個の出力毎のビットが出力される。

【００７４】リファインメント・パス信号７０４及びクリーンアップ・パス信号７０５は、フィードバックカウント信号７０８と共に、マスク７０５へ入力される。カウント信号７０８は、４×４領域内の現在の係数位置、たとえば、０．．．１５である。これらの入力に応答して、マスク７０５は、カウント７０８によって示されるように、既に実行された係数をマスクし、未だ符号化されていない係数だけを組み入れる。たとえば、３個の係数が既に処理されているとき、マスク７０５は、リファインメント出力及びクリーンアップ出力（７０４及び７０５）の夫々の３個の信号ラインをマスクする。

【００７５】マスク７０５は、（一実施例において）一部の信号が１にマスクされた信号７０４及び信号７０５を表現する２個の出力を生成し、優先度符号器７０６へ供給する。マスク７０５の出力は、マスクされたリファインメント標識及びマスクされたクリーンアップ標識（たとえば、信号）である。

【００７６】２個の入力に応じて、優先度符号器７０６は、次のリファインメント・ビット（又は係数）と、有意プロパゲーション・パスに対する次の非クリーンアップ・ビットとを検出し、これらのビットを制御ロジック７０９へ入力する。一実施例において、優先度符号器７０６は、零検出優先度符号器である。その際、優先度符号器７０６は、符号ブロック内のビット（若しくは係数）の現在位置を先行する零のカウントに変換する。一実施例において、これは、以下の真理値表を使用して行なわれる。

【００７７】真理値表

入力	出力
１　×　×　×　×　×	０
０　１　×　×　×　×	１
０　１　１　×　×　×	２

（以下、同様に続く）。

【００７８】マスク７０５、優先度符号器７０６、及び選択ロジック７０７は、決定パスユニット７０２からの出力を受信し、次の非リファインメント係数及び次の非クリーンアップ係数を示す出力と、現在係数に対するパスとを生成する処理ユニットを構成する。

【００７９】入力に応答して、制御ロジック７０９は、リファインメント・次のインデックス、リファインメント・ラン標識、リファインメント・スキップ標識、クリーンアップ・次のインデックス、クリーンアップ・ラン標識、クリーンアップ・スキップ標識、及び、有意プロパゲーション標識を生成する。制御ロジックへの入力は、以下の通りである。

【００８０】次の非リファインビット位置　"R"
次の非クリーンアップビット位置　"C"

```
If R > count then

    refinement run = R-count
    rifinement skip = 0
    cleanup run = 0
    cleanup skip = R-count
    refinement next index = 1
    cleanup next index = 0
    signif prop = 0
Else if C > count
    refinement run = 0
    refinement skip = C-count
    cleanup run = C-count
    cleanup skip = 0
    refinement next index = 0
    cleanup next index = 1
    signif prop = 0
else
    refinement run = 0
    refinement skip = 1
    cleanup run = 0
    cleanup skip = 1
    refinement next index = 0
    cleanup next index = 0
    signif prop = 1            /*  。
```

【００８１】以下の疑似コードは、図７に示された有意＊プロパゲーション・パスロジックの動作を説明する。

```
count = 0
while (count < 16)
    mask = (1 << count)-1
    refinement_masked = refinement | mask
    use priority encoder to find next non-refinement bit
    cleanup_mask = clean_up | mask
    use priority encoder to fine next non-cleanup bit
    if current bit is in significance propagation pass
        process coefficient as significance propagation
        count = count + 1
    else if current bit in refinement pass
        N = "next non-refinement bit" ? count
        process N bits as refinement pass
        count = count + N
    else
        N = "next non-cleanup bit" ? count
        process N bits as cleanup pass
        count = count + N              /*  。
```

【００８２】

【００８３】有意状態は、"1"係数が有意プロパゲーション・パスで符号化されるときにいつでも、ＭＱ復号器（符号化中には、ＭＱ符号器、若しくは、係数値）から更新される。

【００８４】コンテキストモデが１クロックサイクルで動作し、ＭＱ符号器が１クロックサイクルで動作する場合を考えると、二つのクロックサイクルはフィードバックが存在するときに必要とされる。図８には、最悪の状況が起こる可能性のある４×４ブロック上での性能の一例が示されている。８個のコンテキストモデルと、コンポーネントクロックレートの２倍で並列的に動作するＭＱ符号器は、１係数について７個のビットプレーンを復

41

号化できなければならない（８×２／２．２５≒７）。有意プロパゲーション・パス内でスキップが生じない場合、最悪の状況の性能は、最大で１係数当たりについて５．５ビットプレーンまで低下する。何れのパスにおいてもスキップが生じない場合、最悪の状況における性能は、１係数あたりについて高々４ビットプレーンまで低下する。

【００８５】〔ソフトウェアによる有意プロパゲーション・パススキップ〕ソフトウェアの場合、多数のメモリからの並列アクセスは実現不可能である。そのため、一実施例では、符号ブロックは、係数の４×４グループに分割される。グループ毎に、カウントは、有意なビット＊

42

＊数を維持する。このような場合、必要な最大メモリ量は、２５６×５ビットである。リファインメント・パスに存在する全ての係数のブロックのカウントは１６である。カウント０のブロックは、全てクリーンアップでもよく、全部がクリーンアップであるかどうかを調べるためには、近傍を検査するだけでよい。

【００８６】〔クリーンアップ・パス及びリファインメント・パス〕クリーンアップ・パスの場合、アドレッシングは、データに依存し、以下の疑似コードを用いて生成される。クリーンアップ・パス内の次の係数のアドレスｘ，ｙは、入力される。

【００８７】

```
module cleanupAddress(x,y,addrA,addrB,addrC,addrD)
    input[5:0]x;
    input[5:0]y;
    output[6:0]addrA;
    output[6:0]addrB;
    output[6:0]addrC;
    output[6:0]addrD;
    wire[5:0]yp2;
    wire[4:0]ax;
    wire[4:0]bx;
    wire[4:0]cx;
    wire[4:0]dx;
    assign yp2 = y+2;
    assign ax = (x[1:0]==3)?x[5:2]+1:x[5:2];
    assign cx = (x[1:0]==3)?x[5:2]+1:x[5:2];
    assign bx = (x[1:0]==0)?x[5:2]-1:x[5:2];
    assign dx = (x[1:0]==0)?x[5:2]-1:x[5:2];
    assign ay = y[2]?yp2[5:3]+1:yp2[5:3];
    assign by = y[2]?yp2[5:3]+1:yp2[5:3];
    assign cy = yp2[5:3];
    assign dy = yp2[5:3];
    assign addrA = {ay,ax}
    assign addrB = {by,bx}
    assign addrC = {cy,cx}
    assign addrD = {dy,dx}
endmodule                    /*  。
```

【００８８】クリーンアップ・パスに使用されるアドレッシングは、リファインメント・パスにも使用される。しかし、リファインメント・パスの場合、より小さい近※

※傍でも十分に足りる。

【００８９】

```
If (yp2[1:0]==1) or (yp2[1:0]==2) then
    if yp2[2] == 1 then
        just read memories C and D
    else
        just read memories A and B
else
    read memories A,B,C and D                    /*  。
```

【００９０】〔全てのパスに対する逐次的なアドレッシング〕全てのパスに対し逐次的なアドレッシングを行なう場合、二つのメモリを使用するより簡単なメモリ構成が用いられる。図９を参照するに、各４×４領域は、二

つのメモリＡ又はＢのうちの一方に割り当てられる。これにより、１６×１６ブロックに対し要求されるあらゆる並列アクセスが可能になる。１番目の符号ブロックは半分しか使用されない。なぜならば、オフセットは上述のオフセットと類似し、一番上の２行が実際に処理されるデータを収容しない図６に示されるような８×８ブロックを処理するとき、係数の２行分だけが該当するからである。

【００９１】図１０には、有意プロパゲーション・パスのため使用されるメモリ路のメモリ及びレジスタの一実施例が示されている。図１０によると、メモリＡは、アドレスＡに応答してデータ出力を生成する。同様に、メモリＢは、アドレスＢに応答してデータ出力を生成する。２×２クロスバー１００３は、メモリＡ及びＢの出力に接続された入力を有する。クロスバーの一方の出力はレジスタ１００１と、メモリ路の一方の出力と、に接続される。クロスバーのもう一方の出力は、レジスタ１００２と、メモリ路のもう一方の出力と、に接続される。かくして、メモリＡ及びＢの出力は、いずれかのレジスタ１００１及び１００２に保持され、いずれかのメモリ路へ出力される。メモリＡ及びＢから読み出されるデータは４×４領域に対するデータである。レジスタ１＊００１及び１００２は、５×４領域を格納する。レジスタがロードされたとき、最も右側の１×４列は、最も左側の１×４列へ移され、その他の列は、メモリデータ出力から入れられる。クロスバー１００３は、データが１行ずつ処理されるときに、データを出力へ行き来させることにより、メモリＡ及びＢからメモリ路の適切な出力へのデータの出力を制御する。

【００９２】図１１は、図１０のメモリ及びレジスタがコンテキストモデル動作のため適切な領域を設けるため使用される状況を説明する図である。図１１によれば、領域１１０２は、処理されるべき係数の４×４領域である。領域１１０１は、コンテキストモデリングのため使用されたレジスタ１００１及び１００２に格納された５×６領域（５×６領域よりも上及び下の５×１領域は無視される）を表現する。領域１１０３は、メモリからの４×８領域である。領域１１０４は、コンテキストモデリングのため使用されたメモリからの１×６領域である。

【００９３】三つの符号化パスの全てに対するメモリアドレッシング用の疑似コードの一実施例を以下に説明する。

【００９４】

```
address_A_y=0
address_B_y=0
for y=0 to 60 step 4
    address_A_x=0
    address_B_x=0
    clear registers
    read memory A （次に登録されるであろうメモリＡ）
    read memory B （次に登録されるであろうメモリＢ）
    for x=0 to 60 step 4
        address_A_x=x+4
        address_B_x=x+4
        if x < 60 then
            read memory A （次に登録されるであろうメモリＡ）
            read memory B （次に登録されるであろうメモリＢ）
        else
            use "all bits zero" for memory A output
            use "all bits zero" for memory B output
            process 4x4 block of coefficients x...x+3,y...y+3
        if y AND 4==0
            address_A_y=address_A_y+8
        else
            address_B_y=address_B_y+8
```

メモリは、リファインメント・パス及びクリーンアップ・パスのための正しいパスを示すため状態を収容する。状態は、三つの状態（有意プロパゲーション、クリーンアップ及びリファインメント）を識別するため、１係数毎に２ビットである。

【００９５】有意プロパゲーション・パス中に、状態は、１６個の全ての係数に関して並列に、有意係数の場合にはリファインメントに設定され、非有意係数の場合にはクリーンアップに設定される。１６個の係数に関する処理が継続すると共に、有意プロパゲーション・パスに存在する係数の状態は、クリーンアップから有意プロパゲーションへ変更される。状態は、１係数毎に１ビッ

45

トであり、この１ビットをパスビットと呼ぶ。一実施例において、有意状態及びパスビットは、正しいパスを決定するため使用される。表６は、パスビットの用法を例示する。係数１個について１ビットが使用されるので、＊

46

＊メモリ使用量は、ここで説明するラン・スキップカウント法よりも減少する。

【００９６】

【表６】

表６　パスビットの用法

パス	有意状態パスビット 現在	次	コメント
有意プロパゲーション	1，x	1，1	リファインメント・パス内* スタート時の全１６係数に対し並列
	0，0	0，0	クリーンアップ・パス内
		0，1	有意プロパゲーション内の符号" 0"
		1，0	有意プロパゲーション内の符号" 1"
	0，1	0，1	有意プロパゲーション内の符号" 0"
		1，0	有意プロパゲーション内の符号" 1"
リファインメント・パス	1，1	1，1	
クリーンアップ・パス	0，1	0，1	未だ有意でない
		1，x	既に有意になった

一実施例において、表６のリファインメント・パス内*は、４×４ブロックに対する処理のスタート時に全１６係数に関して並列に行なわれる。

【００９７】２×４領域へのアクセスを与えるメモリは、係数毎に有意状態と、パスビットと、サインビットとを備えた４８ビット幅でもよい。

【００９８】図１２は、上記の表６を実現し、現在パスに存在する各係数を検出するため優先度符号器を使用するパス決定ロジックの一実施例のブロック図である。図１２を参照するに、決定パスロジック１２０３は、６×６領域に対する有意状態１２０１と、４×４領域に対するパスビット１２０２と、現在パスを示す現在パス信号（又はその他の標識）１２２０と、を受け取る。パスビット１２０２は、４×４領域内の係数毎の信号（すなわち、１６個の信号）を含む。これらの入力に応答して、決定パスロジック１２０３は、４×４領域に対するパスを指定するため出力を生成する。その際に、４×４領域における各係数に対し、決定パスロジック１２０３は、有意プロパゲーション・パス内の有意プロパゲーション・パスビットを示す信号１２０４、リファインメント・パス内の係数に対するリファインメント・パスビットを示す信号１２０５、又は、クリーンアップ・パス内の係数に対するクリーンアップ・パスを示す信号１２０６をアサートする。

【００９９】選択ロジック１２０７は、現在パス信号１２２０に応答し、標識１２０４～１２０６のうちの一つをマスクロジック１２０８へ出力する。一実施例において、選択ロジック１２０７は、１６×３：１マルチプレクサ（ＭＵＸ）を含む。マスクロジック１２０８は、カウント信号１２０１０に応答して信号を生成し、カウン※ト信号は、現在処理されている係数を指定する。マスク１２０８の出力は、優先度符号器１２０９へ供給され、優先度符号器１２０９は、その信号を制御ロジック１２１２へ出力する。マスクロジック１２０８及び優先度符号器１２０９は、図７に示された同じ名前のロジック及び符号器と同じように動作する。制御ロジック１２１２は、入力に応答して、符号若しくはアイドル状態標識を信号ライン１２１３に生成し、カウント信号１２１０を生成する。

【０１００】次のパスビットロジック１２１１は、優先度符号器１２０９からの（現在処理中の位置を示す）出力と、現在パス信号１２２０と、ＭＱ符号器からの新しい有意状態１２２１と、リファインメント・パス信号１２０５と、を受け取る。リファインメント・パス信号１２０５は、先行の係数が有意であるかどうかを示すことにより、有意状態情報を表現する。現在パス信号１２２０と、新しい有意状態１２２１は、全体として、処理がクリーンアップ・パスで行なわれているかどうかを示す。この入力に応答して、次のパスロジック１２１１は、次のパスビットを生成し、この次のパスビットは、表６における符号０のケースと符号１のケースを区別するために出力として使用される。次のパスビットは、メモリに保持され、次に、パスビット１２０２として使用される。

【０１０１】次に、図１２におけるロジックの動作を疑似コードで説明する。このような機能は、ＭＱ符号器２９０８$_{1-N}$　に組み込まれる。有意状態及びパスビットは、１回目のクリーンアップ・パスを処理する前にクリアされる。

【０１０２】

```
count=0
if significance propagation pass then
    set next pass bit to "1" for all coefficients in refinement pass
```

```
while (count<16)
    if significance propagation pass then
        in_pass=coefficients in significance propagation pass
    else if refinement pass then
        in_pass=coefficients in refinement pass
    else
        in_pass=coefficients in cleanup pass
    mask=(1<<count)-1
    in_pass_masked=in_pass AND (NOT mask)
    use priority encoder to find next coefficients in pass, N
        if next coefficient found
            code coeff N
            if significance propagation pass then
                next pass bit=NOT next significance state
            else next pass bit=pass bit
            count=N+1
        else
            count=16
            idle for coeff N
        next pass bit=pass bit          /*  。
```

【０１０３】上記コードにおいて、変数in_passは、３：１多重化関数の出力である。変数maskは、マスクを表現し、変数in_pass_maskedは、マスクを適用した結果を表現する。変数Nは、パス中の次の係数を表現し、優先度符号器の出力である。変数Nが検出されると、符号体系の制御機能がその後に続く。

【０１０４】上述のコードcode coeff Nは、係数が現在パスに存在するときに、係数を符号化することを意味する。コードidle for coeff Nは、ランを処理中に実行される。

【０１０５】〔２重コンテキスト生成〕コンテキストは、屡々、最後に符号化されたビットに依存し、復号化の際にＭＱ符号器とコンテキストモデルの間に時間的に重大なフィードバックループを生じさせる。このコンテキストモデル遅延を単純な多重化のための時間まで短縮するため、コンテキストモデルは、復号化中の現在ビットが０であるときのための一方のコンテキストと、復号化中の現在ビットが１であるときのためのもう一方のコンテキストの二つのコンテキストを生成することができる。ビットが既知である場合、コンテキストの選択が行なわれる。

【０１０６】図１３は、２重コンテキスト生成ロジックの一実施例のブロック図である。図１３を参照するに、コンテキストモデルは、コンテキスト０、イネーブル０、コンテキスト１、及びイネーブル１を生成する。コンテキストモデルは、２個のコンテキスト０及び１を生成し、両方のコンテキストがマルチプレクサ（ＭＵＸ）１３０２へ送られる。マルチプレクサ１３０２は、信号を受信するよう接続され、コンテキストが有効であるかどうかを示すコンテキスト標識と、ビットを符号化すべきであるかどうかを示すイネーブル標識を生成する。これらの出力は、ＭＱ符号器１３０３の入力へ供給され、ＭＱ符号器はビットを生成する。ＭＱ符号器１３０３からの出力ビットは、ＭＱ符号器１３０３へ出力されるべきコンテキストを選択するためマルチプレクサ１３０２によって使用される。かくして、コンテキストモデルは、現在ビットが０として復号化されたときのコンテキストと、現在ビットが１として符号化されたときの別のコンテキストの２個のコンテキストを生成し、ＭＱ符号器１３０３のための出力ビットは、正しい方のコンテキストを選択する。

【０１０７】クリーンアップ・パス用のランレングス符号化に対し、表７は、ケース毎に、二つの起こり得る次のコンテキストを示し、JPEG 2000標準に記載されたクリーンアップ・パスのフローに従う。ランレングスコンテキストで符号化されたビットの値は、次のコンテキストが次の４係数のグループ用であるか、現在の４係数のグループ用の均一のコンテキストであるかを判定するため使用される。このビットが０である場合、１回の判定で４係数を表現する際に後に続き、後のサイクルのためＭＱ符号器をアイドル状態にさせるランレングス符号化は、速度性能に重大な悪影響を与えない。２番目の均一コンテキスト（ユニフォームＢ）で符号化されたビットの後は、JPEG 2000標準において、（ＸＯＲビット０によって）コンテキスト９で常に直接的に符号化されるサインビットである。ここで、ＸＯＲビットが０であるとき、サインは反転されないことを意味する。

【０１０８】

【表７】

表７
クリーンアップ・パスにおけるランレングス符号化用の２重コンテキスト生成

現在コンテキスト（MQ符号器によって使用中）	"０"に対する次のコンテキスト	"１"に対する次のコンテキスト
ランレングス	ランレングス又は大きさ（又はアイドル状態）	ユニフォームA
ユニフォームA	ユニフォームA	ユニフォームB
ユニフォームB	サイン	サイン

表７中のA及びBは、JPEG 2000標準のセクションD３．４に「均一なコンテキストと共に返される次の２ビット(the next two bits, returned with the UNIFORM context)」のようにに記載された２個のビットを表わす。

【０１０９】有意プロパゲーション及びクリーンアップ符号化パスの場合に、ランレングス符号化を用いない例＊が表８に示されている。大きさビットは符号化されるが、コンテキストは、現在の係数が０であると仮定して次の係数の大きさに対して生成され、或いは、現在の係数に対してサインビットコンテキストが生成される。

【０１１０】

【表８】

表８
有意プロパゲーション及びクリーンアップ符号化パス用の２重コンテキスト生成

現在コンテキスト（MQ符号器によって使用中）	"０"に対する次のコンテキスト	"１"に対する次のコンテキスト
大きさビット	次の係数の大きさ	現在の係数のサイン
サインビット	次の係数の大きさ	次の係数の大きさ

リファインメント・パスの場合、先に符号化されたリファインメント係数はコンテキストに影響を与えない。

【０１１１】〔MQ符号器〕

レートコンテキストを伴うMQ復号器データフロー

図１４Bは、典型的な復号化実現例のブロック図である。図１４Bにおいて、コンテキストモデル１４３０は、メモリ１４３１にコンテキストを供給し、メモリで確率状態が決定される。確率状態は、ロジック１４３２を用いて、算術符号器１４３３のためのＱｅ値に変換され、算術符号器１４３３は、内部Ａ＆Ｃレジスタを更新し、判定結果（ＭＰＳ又はＬＰＳ）を決定する。これらは、全て、典型的に次のコンテキストが決定できる前に行なわれる。殆どのハードウェア実現例の場合、復号化速度は、（コンテキストモデル１４３１へフィードバックする）大きいフィードバックループによって制限される。

【０１１２】これに対し、図１４Aは、初期コンテキストMQ復号器の一実施例のブロック図である。本例の場合、フィードバックループは、復号化動作全体の代わりに、非常に簡単なロジック１４０７を具備する。したがって、殆どの復号及び更新は、下位フィードバックループ１４０１を用いて並列的に行なわれ得る。

【０１１３】図１４Aを参照するに、符号ストリーム１４００が入力され、内部状態１４０１を更新する。一実施例において、内部状態のA及びCレジスタは、JPEG 2000標準の付録Cに記載されているように、現在インターバルを指定する。レジスタAは現在インターバルを指定し、符号レジスタCは、ChighアンドClowレジスタの連結である。

【０１１４】コンテキスト１４０２は、コンテキストモデル１４１０によって与えられる。コンテキスト１４０２は、メモリ１４０３の確率状態１４０４を調べるため使用され、確率状態は、ロジック１４０５によって確率クラス（Ｑｅ値）１４０６に変換される。Ｑｅ値１４０６は、低確率シンボル（ＬＰＳ）の現在推定値を表現する。Ｑｅ値１４０６は、ロジック１４０７を用いて、出力判定結果１４０８を生成するため、MQ符号器の内部状態のJPEG 2000標準の図Ｃ－１５に記載されているAレジスタ値及びCレジスタ値と比較される。出力判定結果は、より確率が高い高確率シンボル（ＭＰＳ）若しくはＬＰＳである。出力判定結果１４０８は、コンテキストモデル１４１０へ入力される。一実施例において、Ｑｅ値及び内部状態に関する演算は、１６ビット演算を必要とする。これらのブロックの演算は、JPEG 2000標準のセクションＣ．３．２に記載されているような判定結果の復号化を実施する。

【０１１５】図１５は、後期コンテキストMQ復号器の一実施例のブロック図である。図１５では、１６ビット処理がコンテキストモデルフィードバックループから除去されている。符号ストリーム１６０１は、更新ロジック１５０３への入力として受信され、更新ロジックは、内部状態を更新し、現在インターバルを指定するAレジスタ及びCレジスタを含む。新しいAレジスタ値及びCレジスタ値、並びに、符号ストリームは、ロジック１５０４へ入力され、ロジック１５０４は、以下で説明するような二つのｐクラス(pclass)、すなわち、ｐクラス１５０９及びｐクラス１５１０を生成する。二つのｐクラスは比較ロジック１５１１及び１５１２へ入力される。

【０１１６】コンテキストモデル１５２０は、コンテキスト１５０２を生成する。コンテキスト１５０２は、メモリ１５０５の確率状態１５０６を調べるため使用される。一実施例において、メモリ１５０５は、ルックアップテーブルを含む。確率状態１５０６を確認することによって、Ｑｅ値を決定できるようになる。メモリ１５０５から出力された確率状態１５０６は、ロジック１５０７によって、確率クラス（インデックス）１５０８へ変換される。

【０１１７】比較ロジック１５１１は、ｐクラス１５０９が確率クラスインデックス１５０８よりも大きいかどうかを判定し、比較ロジック１５１２は、確率クラスインデックス１５０８がｐクラス１５１０よりも大きいかどうかを判定するため比較を行なう。両方の比較ロジック１５１１及び１５１２の結果は、両方の比較結果が真である場合に、判定結果が出力されるように、ＡＮＤゲート１５１３へ入力される。この判定結果は、ＭＰＳ若しくはＬＰＳである。かくして、コンテキスト１５０２は、（JPEG 2000では、Ｑｅ値に対し３２個の値を取り得るので）５ビット確率クラスインデックス１５０８へ変換される。内部状態は、二つの５ビット確率クラスインデックスを生成するため使用される。コンテキストに対応したインデックスが状態から生成された二つのインデックスの外側にあるとき、判定結果はＭＰＳであり、さもなければ、判定結果はＬＰＳである（すなわち、二つのインデックスの内側にある）。

【０１１８】図１５の実施例の重要な利点は、内部状態更新が、図１４Ｂに示されるように逐次的ではなく、次の確率クラス（インデックス）１５０８の生成と並行して行なわれることである。また、二つの確率クラスは、５ビットだけがｐクラスインデックスと比較されるので、演算は非常に簡単化される。

【０１１９】図１５のロジック１５０４は、図１６Ａに示される情報を作成する。Ａレジスタ及びＢレジスタに値が与えられると、ロジック１５０４は、ｐクラスに対する二つの分割点が何であるかを決定し、次に、符号が分割点の間にあるか、又は、外側にあるかを判定する。これは，並列に行なってもよい。

【０１２０】図１６Ａは、確率クラスインデックスの比較がどのように行なわれるかを示す図である。図１６Ａにおいて、ｐクラス０は、殆どのインターバルがＭＰＳへ向けられる非常に歪んだケースである。ｐクラス１からｐクラス４に対し、この歪みは小さくなり、ＭＰＳインターバルは縮小する。ｐクラス５は、ＭＰＳに５０％付近の確率に対して生じる条件付き交換を与える。既知状態は、一部の確率クラスに対してＭＰＳであり、他の確率クラスに対してＬＰＳである符号ストリーム値（符号）を有する。確率クラスは順序付けられているので、符号がＭＰＳであるか、ＬＰＳであるかを判定するためには、２回の比較だけで十分である。すなわち、図１６Ａにおいて、符号のロケーションが既知状態で与えられた場合、判定結果は、ｐクラス０～３に対してＭＰＳであり、ｐクラス４に対しては常にＬＰＳであり、ｐクラス５に対してもＭＰＳである。各確率クラスに対しＭＰＳであるか、ＬＰＳであるかを見つけるのではなく、（ｐクラス３とｐクラス４の間のブレークポイント、及び、ｐクラス４とｐクラス５の間のブレークポイントの）二つのブレークポイントだけを判定すればよい。したがって、ＱＥ値が与えられたとき（確率クラス／インデックスが既知であるとき）、ブレークポイントが存在する空間に実際に現れる確率クラスが何であるかが決定される。

【０１２１】ハードウェアにおける同様の方法は、起こり得るＱｅ値毎にＭＰＳかＬＰＳかを判定し、次に、その結果を多重化するために使用される。たとえば、図１６Ｂには、多数の入力を有するマルチプレクサ１６１０が示され、各入力はｐクラスと関連づけられ、符号に応じて、ＭＰＳ若しくはＬＰＳのいずれかを出力として生ずる。

【０１２２】〔ＭＱ符号器による多重ビット復号化〕多数のＭＰＳは、正規化を必要とするＭＰＳが存在しない限り、又は、最後のＭＰＳだけが正規化を必要とする限り、（同じＰクラスを継続して使用するため）同時に復号化可能である。図１７は、多重ＭＰＳ復号化のためのインターバルを示す。標準的に、ＡレジスタとＣレジスタによって指定されたインターバルに対する符号ストリームの場所と、Ｑｅ値との差が２以上であるならば、多数のＭＰＳを符号化することが可能である。インターバルサイズがＱｅ値によって分割され、復号器が同じコンテキストに留まる場合、すなわち、同じ確率クラスに留まる場合、多数のＭＰＳを同時に復号化することが可能である。たとえば、符号ストリームを調べ、処理されるのが１６ビットであることがわかったとき、ＡレジスタとＣレジスタによって指定されたインターバル内の符号ストリームの場所は、Ｑｅ値の倍数だけ離間し、同じコンテキスト、つまり、同じ確率クラスが多数のサイクルのデータを処理するために使用される予定であることを示し、多数のＭＰＳが同時に復号化される。つまり、

｛（ＡレジスタとＢレジスタによって指定されたインターバル）－（符号ストリームの場所）｝／Ｑｅ

が判定され、この値よりも大きい最小の整数に丸められたとき、その結果は、同時に復号化されるＭＰＳの個数を表わす。この計算は周知のハードウェアによって実行される。

【０１２３】〔５，３フィルタの典型的な実現形態〕一実施例において。可逆及び不可逆５，３ウェーブレットフィルタが使用される。ここで、５，３は、ウェーブレットフィルタ内のタップ数、すなわち、カーネルの基底関数サポートにおける非零（連続）値の数である。可逆とは、順変換と逆変換を実行すると、入力と厳密に同じ

数が出力で得られることを意味する。入力精度に関して適度で予測可能な精度の増加が中間数学的項のため要求される。すなわち、数学的精度によって規則歪みは導入されない。不可逆とは、数学的歪みの無いこと（正確な再生）を保証するために、非常に高い精度が要求されることを意味する。しかし、実際上、不可逆フィルタは、殆どの場合に、規則的数学精度歪みを圧倒する歪みを生ずる量子化と組み合わされる。

【０１２４】図２４には、順変換フィルタの一実施例が示されている。図２４によると、高域通過（ハイパス）フィルタ２４０２は、ラインバッファ２４０１からラインｘ０ ｘ１ 及び最終ｘ０を受け取り、低域通過（ローパス）フィルタ２４０４の一つの入力へ供給され、ラインバッファ２４０３に保持される出力を生成する。ラインバッファ２４０１及び２４０３は、タイル幅を有する１本のラインを保持する。ローパスフィルタ２４０４は、ハイパスフィルタ２４０２の前のサイクルの出力、すなわち、ラインバッファ２４０３からの出力を、現在のｘ０ラインと共に受け取り、出力を生成する。過去の２クロックサイクルに対するローパスフィルタ２４０４の出力は、遅延器２４０５及び２４０６によって遅延され、１サイクル前のフィルタ出力と、２サイクル前のフィルタ出力を与える。

【０１２５】ハイパスフィルタ２４０２の過去の出力は、遅延器２４０７及び２４０８によって遅延され、ハイパスフィルタ２４０２の現在出力と、ハイパスフィルタ２４０２の最後の２個の出力が、ハイパスフィルタ２４１３へ供給される。ハイパスフィルタ２４１３の出力は、ＨＨサブバンドにおける係数であり、ハイパスフィルタ２４０２の２サイクル前の過去の出力、すなわち、遅延器２４０８からの出力、及び、ハイパスフィルタ２４１３の前の出力と共に、ローパスフィルタ２４１５へ供給される。ローパスフィルタ２４１５の出力はＨＬサブバンドからの出力である。

【０１２６】ローパスフィルタ２４０４の出力は、遅延器２４０５及び２４０６の出力と共に、ハイパスフィルタ２４０９へ供給される。ハイパスフィルタ２４０９の出力はＬＨサブバンドである。

【０１２７】ハイパスフィルタ２４０９の出力は、遅延器２４０６の出力、及び、遅延器２４１０によって遅延されたハイパスフィルタ２４０９の前の出力と共に、ローパスフィルタ２４１１の入力へ供給される。ローパスフィルタ２４１１の出力はＬＬサブバンドである。ローパスフィルタ２４１１の出力であるＬＬサブバンドがラインバッファ２４１２へ供給されるとき、ラインバッファ２４１２の出力は、ローパスフィルタ２４１１の出力と共に、ウェーブレット変換の次のレベルへの入力を表わす。ウェーブレット変換の次のレベルは、図２４に示されたウェーブレット変換を縦続接続したものである。

【０１２８】図２５Ａは、ここで説明される変換（たとえば、上述の５，３変換）で使用されるようなローパスフィルタの一実施例の説明図である。ローパスフィルタは、

$$-x_0+2x_1-x_2$$

に従う関数に基づいて出力を生成するよう設計される。可逆変換の場合、ローパスフィルタは、次式

【０１２９】

【数１】

$$x_1-\left\lfloor\frac{x_0+x_2}{2}\right\rfloor$$

に従って動作する。

【０１３０】図２５Ａを参照すると、加算器２５０１は、最後のｘ０ラインを現在のｘ０ラインと加算する。最下位ビット出力は、図２５Ｂのハイパスフィルタの出力を表わし、不可逆変換のため使用される。残りのビットは、減算器２５０２へ供給され、最上位ビットを表現する出力を生成するためｘ１入力から減算される。これらの最上位ビットは、すべて可逆変換の場合に必要である。逆ウェーブレット変換の場合、図２５Ａのフィルタを、逆変換で奇数（ハイパス）フィルタとして使用するための逆ウェーブレットフィルタに切り替えるため、減算器２５０２は加算器で置換される。このような一例は、図２５Ｂのハイパスフィルタに示されている。

【０１３１】図２６Ａは、ここで説明する変換に使用されるハイパスフィルタの一実施例の説明図である。不可逆変換の場合、ハイパスフィルタは、以下の式、

$$4x_1-x_0-x_2$$

に従って動作する。

【０１３２】可逆変換の場合、ハイパスフィルタは、次式、

【０１３３】

【数２】

$$x_1-\left\lfloor\frac{x_0+x_2+2}{4}\right\rfloor$$

に従って動作する。

【０１３４】図２６Ａを参照すると、加算器２６０１は、最後のｘ０ラインの可逆側又は不可逆側のいずれかを現在のｘ０ラインに加算する。加算器２６０１の出力は、加算器２６０３を用いて丸め項に加算される。丸め項は、可逆側の場合には２であり、不可逆側の場合には０であり、マルチプレクサ２６０２によって供給される。加算器２６０３の下位２ビットを除く出力は、加算器２６０４を用いて、ｘ１ラインに加算され、可逆側出力を生ずる。加算器２６０３の出力の下位２ビット、及び、加算器２６０４の出力は、不可逆側出力を表わす。

【０１３５】マルチプレクサ２６０２を用いることによって、可逆側と不可逆側を切り替えるために、両方の機能のための完全に別個のハードウェアを必要とすることなく、或いは、可逆側丸めが不可逆側出力に影響を与えることを要することなく、簡単な切り替えを実現できる

ようになる。

【０１３６】逆ウェーブレット変換の場合に、図２６Ａのフィルタを、逆変換中の偶数（ローパスフィルタ）として使用するための逆ウェーブレットフィルタに切り替えるため、加算器２６０４は減算器で置換される。このような一例は、図２６Ｂのローパスフィルタに示されている。

【０１３７】図２７は、図２４における変換の代替的な実施例を示す図であり、イメージ境界でミラーイメージ処理を実行するためのマルチプレクサを含む。マルチプレクサは、マルチプレクサ２７０１から２７１２により構成される。たとえば、マルチプレクサ２７０１は、バッファ２４０１にラインが存在しないとき（たとえば、タイルの一番上）、境界において最後のｘ０ラインの代わりにｘ０ラインを使用する。マルチプレクサ２７０２は、タイルの一番下に到達したときに入力されるべき更なるｘ０ラインが存在しない場合、ラインバッファが他の入力をローパスフィルタ２４０４へ供給できるようにする。同様に、マルチプレクサ２７０３は、バッファ２４０３にラインが存在しない場合、ハイパスフィルタ２４０２の出力がローパスフィルタ２４０２への入力として利用できるようにさせる。マルチプレクサ２７０４は、ハイパスフィルタ２４０２からの出力が無い場合、ローパスフィルタ２４０４への入力を、ラインバッファ２４０３から供給できるようにする。マルチプレクサ２７０５及び２７０６は、ローパスフィルタ２４０４への出力及び遅延器２４０６からの出力が利用できないとき、ハイパスフィルタ２４０９への入力に、それぞれ、遅延器２４０６の出力、及び、ローパスフィルタ２４０４の出力を供給できるようにする。マルチプレクサ２７０９及び２７０１と、マルチプレクサ２７０７及び２７０８と、マルチプレクサ２７１１及び２７１２についても、マルチプレクサ２７０５及び２７０６と同様のことが成り立つ。

【０１３８】図２８は、逆５，３変換の一実施例のブロック図である。図２８を参照するに、偶数フィルタ２８１５は、ＬＬ係数、ＨＬ係数、及び、遅延器２８８０１からの前のサイクルのＨＬ係数を、受け取るように接続されている。偶数フィルタ２８１５の出力は、偶数フィルタ２８１１の一入力と、遅延器２８０２の一入力と、奇数フィルタ２８０３の一入力とへ接続される。奇数フィルタ２８０３の別の入力には、遅延器２８０１を介して前のサイクルのＨＬ係数が供給され、遅延器２８０２を介して、偶数フィルタ２８１５の前のサイクルの出力が供給される。奇数フィルタ２８０３の出力は偶数フィルタ２８１０の一入力へ接続される。

【０１３９】フィルタ２８０５は、現在のＨＨ係数及びＬＨ係数と、遅延器２８０４からの前のサイクルにおけるＨＨ係数とを受け取るように接続されているので、上記の構成と同じような構成がＬＨ係数及びＨＨ係数に関しても存在する。偶数フィルタ２８０５の出力は、偶数フィルタ２８１１の一入力と、遅延器２８０６の入力と、奇数フィルタ２８０７の一入力とへ接続される。奇数フィルタ２８０７の別の入力には、前のサイクルのＨＬ係数（すなわち、遅延器２８０４の出力）と、偶数フィルタ２８０５の前のサイクルの出力（すなわち、遅延器２８０６の出力）と、が供給される。奇数フィルタ２８０７の出力は偶数フィルタ２８１０の一入力へ接続される。

【０１４０】偶数フィルタ２８０５の出力及び奇数フィルタ２８０７の出力は、それぞれ、ラインバッファ２８０８及びラインバッファ２８０９へ供給され、保持されることに注意する必要がある。ラインバッファ２８０８及び２８０９のサイズは、タイル幅の１／２に一致する。ラインバッファ２８０８の出力は、偶数フィルタ２８１１の別の入力と、奇数フィルタ２８１５の一つの入力とに接続される。ラインバッファ２８０８の出力は、偶数フィルタ２８１０の一つの入力と、奇数フィルタ２８１４の一つの入力とに接続される。

【０１４１】偶数フィルタ２８１０の出力は、出力される画像データの”Ｃ”部分であり、ラインバッファ２８１２に保持され、奇数フィルタ２８１４の入力に供給される。一実施例において、ラインバッファ２８１２のサイズは、タイル幅の１／４に一致する。この入力に応答して、奇数フィルタ２８１４は、画像データの”Ａ”部分に対応するデータを生成する。

【０１４２】偶数フィルタ２８１１の出力は、画像データの”Ｄ”部分に対応し、奇数フィルタ２８１５の一つの入力へ供給され、ラインバッファ２８１３に保持される。一実施例において、ラインバッファ２８１３のサイズは、タイル幅の１／４である。ラインバッファ２８１３の出力は奇数フィルタ２８１５の別の入力へ接続される。奇数フィルタ２８１５の出力は、画像データの”Ｂ”部分に対応する。

【０１４３】〔その他の並列方式実施技術〕

符号ブロックの並列用符号器への割り付け

ハードウェアの実施例の場合、同じタイル内で並列に多数の符号ブロックを符号化することが有効である。図２１は、多数のＭＱ符号器を含む符号器の一実施例のメモリ使用量説明図である。各ＭＱ符号器は、関連したコンテキストモデルを具備し、多数の符号ブロックを処理するため使用される。

【０１４４】図２１を参照すると、各ＭＱ符号器は、メモリ（たとえば、別々のメモリ、単一若しくは多数のメモリの一部）が割り付けられる。一実施例において、割り付けられたメモリのある部分は、符号化データを保持し、長さ、零ビットプレーン、及び、符号化パスはメモリの別の部分に保持される。

【０１４５】図１８～２０には、１２８×１２８タイル、６４×６４符号ブロック、及び、夫々の変換レベル

に関する符号ブロックの並列ユニットへの割当が示されている。割当は、並列符号器毎に実行される符号化の量が均衡するに行なわれる。一実施例において、符号ブロックの割当は、各ＭＱ符号器が、できるだけ可能な範囲で、上位レベル係数と下位レベル係数の間で均衡を保ちながら略同数の係数を符号化するように、行なわれる。これ以外の構成も実現可能である。

【０１４６】図１８Ａ、Ｂ及びＣには、それぞれ、４台、６台及び８台のＭＱ符号器が並列に使用された場合の４：４：４データに関する符号ブロックの割当の実施例が示されている。図１８Ｃにおいて、８個のユニットが並列である場合、並列ユニット”Ｈ”（１ＨＨクロミナンスサブバンド）に割り当てられた符号ブロックは、屡々、重く量子化されるので（符号化すべき非零ビットプレーンが殆ど存在しないので）、このユニットは、単位時間当たりに処理できる係数の数が他のユニットよりも多くなる可能性が高い。

【０１４７】図１９Ａ、Ｂ及びＣには、それぞれ、４台、６台及び８台のＭＱ符号器が並列に使用された場合の４：２：２データに関する符号ブロックの割当の実施例が示されている。

【０１４８】図２０Ａ、Ｂ及びＣには、それぞれ、４台、６台及び８台のＭＱ符号器が並列に使用された場合の４：１：１データに関する符号ブロックの割当の実施例が示されている。図２０Ｃにおいて、８個のユニットが並列である場合、ユニットＣ、Ｄ及びＥが単位時間当たりに処理できる係数の数は、他のユニットよりも多くなることが予想される。

【０１４９】図２９の符号器は、上述の符号化を実行するため使用される。たとえば、ビットモデリングＭＱ符号器2908_{1-N}　のＮ台のＭＱ符号器のうちの各符号器は、図１８乃至２０に示されたＡ～Ｈのいずれかに割り当てられる。

【０１５０】図１８乃至２０に関して、偶数台のＭＱ符号器が並列にされているが、並列にされるＭＱ符号器の台数は奇数台でも構わない。

【０１５１】係数をハードウェアに格納するためのメモリ節約

無損失の復号化ではないときに、係数を保持するメモリ使用量を削減するために、零ビットプレーン変換を利用することが可能である。ハードウェアが係数毎にＮ個のビットプレーンを保持できる場合、復号化は、Ｎ個のビットプレーンが復号化された後に終了する。後続のビットプレーンは量子化（丸め処理）を行なうことが可能である。

【０１５２】図２２Ａは、符号化中に、係数毎に、メモリの有限個のビットプレーンを使用する例が示されている。たとえば、メモリの８個のビットプレーン（Ｎ＝８）は、標準的な表現形式で、係数を１６ビットによって符号化するため使用され得る。これらの係数は、（Ｌサブバンド係数は量子化されない）ＬＬサブバンド以外のサブバンドの一部であり、ウェーブレット変換を画像データに適用した結果として生成される。一実施例において、ウェーブレット変換は、５，３ウェーブレット変換を含む。ウェーブレット変換は、ＬＬサブバンド、ＨＨサブバンド、ＬＨサブバンド及びＨＬサブバンドを並列に生成するため、並行動作する多数の５，３ウェーブレット変換により構成しても構わない。ウェーブレット変換からの係数を保持するメモリは、係数ビットに基づく符号化を実行するため、コンテキストモデルによってアクセスされる。

【０１５３】符号化中に、係数は、零ビットプレーンの数が基地になる前に保存される。カウンタは、上位のビットプレーン８～１５に対する初期零の数をカウントする。ビットプレーン８～１５が全て零である限り、メモリは、対応したビットプレーン０～７に対する情報（大きさ）を保持する。ビットプレーン８～１５に１が出現した場合、対応したカウンタは停止し、メモリは対応したビットプレーン８～１５の情報を保持する。符号ブロックの符号化の最後に、カウンタがビットプレーン８～１５に対し全て零を指定し、対応したビットプレーン０～７は、カウンタがメモリの最後に値を格納した場合にメモリに収容するか、又は、カウンタはメモリ内のビットプレーン８～１５に対する開始アドレスを指定し、対応したビットプレーン０～７を丸め処理（量子化）する必要がある。かくして、カウントは、サイドバンド情報として作用し、行の始まりからカウントによって指定された行内の位置までメモリアレイに保持された情報が必要の無いデータになったことを示す。

【０１５４】メモリの別々のビットプレーンはサイン情報を保持するため使用されるか、又は、サイン情報は有意状態と共に保持される。

【０１５５】他の実施例において、小量のメモリが、カウンタの代わりに、可変長（ＶＬ）符号情報（たとえば、ランレングス符号）のため使用される。これにより、少数の”１”ビットを含むビットプレーンを、ビットプレーン毎にメモリの一部分に格納することが可能になる。ビットをメモリに格納した後、コンテキストモデルは，ビットを取得するためメモリにアクセスする。しかし、各行は、潜在的に、量子化されるべきデータを含んでいるので、コンテキストモデルによってアクセスされ、使用される必要が無い。図２２Ｂは、メモリへのアクセスを制御するための制御ロジックの一実施例のブロック図である。このロジックは、メモリをアクセスするコンテキストモデルと協働して、或いは、コンテキストモデルの一部として動作する。

【０１５６】図２２Ｂを参照するに、アドレスａｄｄｒは、ビットを生成するメモリアレイ２２０１にアクセスする。アドレス、及び、アドレスを収容したメモリの行と関連したカウンタ値は、比較ロジック２２１０へ供給

される。比較ロジック２２１０において、アドレスが行に対するカウンタ値以上であると判定された場合、メモリアレイ２２０１からの１ビット出力が生成され、そうでなければ、零が出力される。

【０１５７】図２３は、ＶＬ符号からのメモリの一部分と、係数を格納するメモリアレイとを示す図である。ＶＬ符号は、行内で次の”１”が現れるまでスキップするビット数を示すことによって１ビットの有無を指定するため使用される。かくして、このＶＬ符号は、アクセスロジックが次のビットプレーンの場所を知ることができるように２個のカウントを指定するため作成される。他のＶＬ符号は、３個以上のカウントを与えるため使用される。ＶＬ符号を使用することによって、典型的に、満杯よりも一つ少ないメモリのビットプレーンを使用できるようになる。小容量メモリのサイズが（１ビットプレーン当たり）符号ブロックのサイズの１／３２であるならば、Ｒ２［8］符号又はＲ２［7］符号を使用してもよい。Ｒ２［8］、Ｒ２［6］、及びＲ２［7］符号の詳細な情報は、本願の譲受人に譲受された、米国特許第5,381,145号、発明の名称"Method and Apparatus for Parallel Decoding and Encoding of Data"、1995年1月10日発行に記載されている。

【０１５８】変換及びコンテキストモデル／ＭＱ符号器の並列動作が望ましいビデオの場合、２個のメモリバンクが必要になる。静止画像アプリケーションの場合、変換及びコンテキストモデル／ＭＱ符号器の逐次的動作のために１個のメモリバンクで十分である。

【０１５９】上記のメモリ節約技術は行に関して説明さ＊れているが、たとえば、列、ブロック、ページ、領域などの任意のメモリ領域を使用することができる。或いは、別個のメモリを使用してもよい。

【０１６０】パケットヘッダ処理

たとえば、JPEG 2000符号ストリーム（又は、ビットストリーム）のような符号ストリームを作成するため、パケットヘッダが作成される。一実施例において、この情報は、任意の個数の符号ブロックを取り扱うためタグ木構造を伴う。ある種の状況では、制限された個数の符号ブロックを備えたタイル用のタイルヘッダが作成される。たとえば、タイルが、４個の１２８×１２８サブバンドを含み、各サブバンドが６４×６４符号ブロックに分割される場合、一体として符号化される符号ブロックの個数は４個である。パケットヘッダは、特定の符号ブロックに対するデータが存在するかどうか、データが存在する場合の零ビットプレーンの個数、符号化データの長さ、及び、データに含まれる符号化用パスの個数を指定する。

【０１６１】以下の表９は、２×２符号ブロック及び１レイヤを含むパケット用のパケット構造の一実施例を示す。表９を参照するに、タグ木構造は、２レベルの深さしかない。記号”ｚ”は、最高レベルの零ビットプレーンのタグ木構造情報の場所を示し、記号”＿”で示された場所は、残りの零ビットプレーン、符号化用パス及び長さ情報の置かれる場所を示す。

【０１６２】

【表９】

表９　２×符号ブロック及び１レイヤ用の包含情報

包含情報	符号
0000	0*
0001	110001z_
0010	11001z_0
0011	11001z_1_
0100	1101z_00
0101	1101z_01_
0110	1101z_1_0
0111	1101z_1_1_

包含情報	符号
1000	111z_000
1001	111z_001_
1010	111z_01_0
1011	111z_01_1_
1100	111z_1_00
1101	111z_1_01_
1110	111z_1_1_0
1111	111z_1_1_1_

表中、＊は、０又は１０又は１１０００００を表わす。

【０１６３】一実施例において、符号110000は、実現例の便宜上、符号ブロックが包含されない場合に使用される。

【０１６４】制限された数の符号ブロック及び単一レイヤを備えたタイル用のパケットヘッダを書き込む手続の一実施例は、以下のような初期化：サブバンド毎に、零ビットプレーン最小値ＭＺＰに最大値を設定から始まる。

【０１６５】一実施例において、ＭＺＰの最大値は、１５ビットプレーンまでの場合、0xFであり、３１ビットプレーンまでの場合、0x1Fである。使用される値が大きくなると共に、実施例で取り扱い得るビットプレーンの数が増加する。

【０１６６】次に、パケット内で各符号ブロックの係数を符号化するとき、

Save included or not bit（包含又は包含されないビットを保存）

Save number of zero bitplanes（零ビットプレーンの個数を保存）

If zero bitplanes less than MZP then MZP = zero bitplanee（零ビットプレーンがＭＺＰ未満であるならば、ＭＺＰ＝零ビットプレーン）

Save number of coding passes（符号化用パスの個数を保存）

Save Length（長さを保存）

を実行する。

【０１６７】Save included or not bitは、（量子化後

61

の）全ての係数が零である場合に、セットされ、符号ブロックが包含されないことを指定する。最終的に、タイル若しくはサブバンドの情報が処理された後、パケット＊

```
write "1"
for each subband
    write "1"
    first_flag = 1
    for each code-block
        if not included then
            write "0"
        else
            write "1"
            if first_flag then
                write MZP in tag tree format
                first_flag = 0
            write zero bitplanes ? MZP in tag tree format
            write coding passes
            determine minimum Lblock value
            write Lblock
            write length
```

ここで、Lblockは、JPEG 2000標準のセクションB.10.7.1に規定されている。

【０１６９】パケットヘッダは、少なくとも１バイトであり、JPEG 2000準拠式復号器は、書き込まれた情報を認識できることに注意する必要がある。

【０１７０】多数のレイヤが存在する場合、ＭＺＰ変数の初期化は、１レイヤのバイト同様に行なわれる。各符号ブロックの符号化中に、包含又は非包含の指標、符号化用パスの個数、及び、長さがレイヤ毎に保存される。さらに、以下の初期化処理が好ましい。

【０１７１】first_flag = 1

※

```
write "1"
for each subband
    if layer 0 then write "1"
    for each code-block
        if not included then
            write "0"
        else
            write "1"
            if code-block not already included then
                if first_flag then
                    write MZP in tag tree format
                    first_flag = 0
                write zero bitplanes ? MZP in tag tree format
                set already included
            write coding passes
            determine minimum Lblock value
            write Lblock
            write length
```

62

＊ヘッダは以下のように記述される。

【０１６８】

※initialize Lblock for each code-block
initialize already included for each code-block to false

一実施例において、Lblockは３へ初期化される。"already included"が真であるとき、一部の先行レイヤがデータを符号化したことを意味する（すなわち、符号ブロックが過去に出現している）。

【０１７２】レイヤ毎にパケットを書き込むため、以下の手続が使用される。

【０１７３】

"already included"情報は、各符号ブロックに対し別個のビットでも構わない。或いは、零ビットプレーンの使用されない値を"already included"を指定するため使用してもよい。たとえば、１４個のビットプレーンが存在する場合、零ビットプレーンを１５(0xF)にセットすることによって、"already included"を示すことが可能である。

【０１７４】"０"パケットを使用しない圧縮符号化データ

JPEG 2000の場合、パケットヘッダはバイト単位に丸められる。一部のケースでは、しかし、パケットヘッダは、唯一の零ビット、若しくは、パケットヘッダをバイト境界に収容するために必要なビット数よりも少ない多数のビットを格納する。パケットヘッダは、一般的に、パッディングによってバイトに丸められる。また、パケットヘッダ表現形式は、固有ではなく、典型的には、できるだけ短い表現形式の使用が望まれるが、一実施例では、使用される過剰ビットがパッディングによって埋められるであろうビット場所の代わりをするならば、考えられる最短の形式ではない表現形式が使用される。これは、過剰ビットで符号化された情報がタイルコンポーネントレベル分割における次のパケットに関する情報を示す場合に、特に有効である。

【０１７５】たとえば、単一のサブバンドが存在し、２×２ブロックが包含される場合、零パケットを出力する。しかし、同じ空間量で、パケットに何かが存在し、タグ木構造の一番上のレベルには何も包含されていない、ということを示すために零を出力してもよい。或いは、タグ木構造に何かが存在し、それが０００００である（すなわち、４箇所に個別には何も存在しない）ことを示すことも可能である。かくして、これらの過剰ビットは、より多くのタグ木構造情報を与えるため使用される。このタグ木構造情報は、後でパケットヘッダに出現し、必ず繰り上げられるであろう情報である。ビットを先行のパケットヘッダへ繰り上げることにより、符号ストリーム全体のサイズを１バイトずつ（或いは、それ以上）縮小することが可能である。

【０１７６】以上の説明から、本発明の多数の代替及び変更が当業者に明らかになるであろう。しかし、例示のために解説され、図示された具体的な実施例は、本発明を制限することを意図していないことに注意する必要がある。したがって、多数の実施例の詳細の説明は、本発明に不可欠であると考えられる事項だけが記載された請求項に挙げられた事項の範囲を制限することを意図したものではない。

【図面の簡単な説明】

【図１】JPEG2000復号化体系のブロック図である。

【図２Ａ】係数毎にサブビットプレーンパスが識別され、各符号化パスのための処理の順序を示すラベルが付けられた係数の８×８符号ブロックの一例を示す図である。

【図２Ｂ】可変長ラン・スキップカウント用メモリの説明図である。

【図３】（Ａ）～（Ｄ）は、コンテキストモデルの一実施例に対する近傍係数とメモリ構成を示す図である。

【図４】１６×１６符号ブロックのランダムアクセス用の有意メモリ構成の一実施例の説明図である。

【図５】ランダムアクセス用の有意プロパゲーション・パスに使用されるメモリ及びレジスタを示す図である。

【図６】メモリからランダムアクセス用レジスタに格納された有意状態を示す図である。

【図７】有意プロパゲーション・パスロジックの一実施例のブロック図である。

【図８】４×４ブロック上のコンテキストモデルの一実施例の性能の一例を示す図である。

【図９】１６×１６符号ブロックのシーケンシャルアクセス用有意メモリの構成の一実施例の説明図である。

【図１０】有意プロパゲーション・パスのため使用されるメモリ及びレジスタの一実施例の説明図である。

【図１１】コンテキストモデル動作用の適切な領域を与えるためのメモリ及びレジスタの使用法の説明図である。

【図１２】パス決定ロジックの一実施例のブロック図である。

【図１３】２重コンテキスト生成ロジックの一実施例のブロック図である。

【図１４Ａ】初期コンテキスト型ＭＱ符号器の一実施例のブロック図である。

【図１４Ｂ】典型的な復号化実現方式の一実施例の説明図である。

【図１５】後期コンテキスト型ＭＱ符号器の一実施例のブロック図である。

【図１６Ａ】確率クラスインデックスの比較の動作状態の説明図である。

【図１６Ｂ】各Ｑｅ値に対するＭＰＳ又はＬＰＳを決定するマルチプレクサのブロック図である。

【図１７】多重ＭＰＳ復号化のためのインターバルの説明図である。

【図１８】４：４：４データに対する並列的な符号ブロックの割当の一実施例の説明図である。

【図１９】４：２：２データに対する並列的な符号ブロックの割当の一実施例の説明図である。

【図２０】４：１：１データに対する並列的な符号ブロックの割当の他の一実施例の説明図である。

【図２１】個々に関連性コンテキストモデルを有する多数のＭＱ符号器を含む符号器の一実施例のメモリの構成図である。

【図２２Ａ】符号化中に各係数に対し制限された数のメモリのビットプレーンを使用する例の説明図である。

【図２２Ｂ】メモリへのアクセスを制御するための制御

65

ロジックの一実施例のブロック図である。

【図２３】カウンタの代わりに、可変長（ＶＬ）符号情報用の小容量のメモリを使用する例の説明図である。

【図２４】順変換の一実施例のブロック図である。

【図２５Ａ】ローパスフィルタの一実施例のブロック図である。

【図２５Ｂ】ハイパスフィルタの一実施例のブロック図である。

【図２６Ａ】ハイパスフィルタの一実施例のブロック図である。

【図２６Ｂ】ローパスフィルタの一実施例のブロック図である。

【図２７】順変換の他の一実施例のブロック図である。

【図２８】逆変換の一実施例のブロック図である。

【図２９】符号器／復号器の一実施例のブロック図である。

66

＊【図３０】ランカウント及びスキップカウントの両方を有する１６ビット語の一例の説明図である。

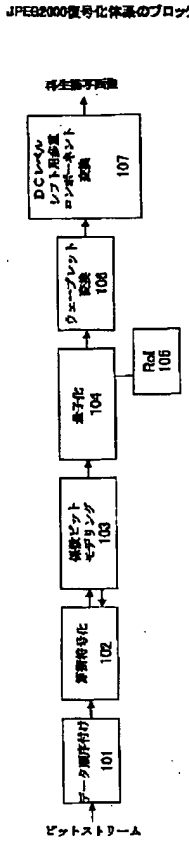
【図３１】符号化パスを決定する有意状態ビットの典型的な８×８領域の説明図である。

【図３２】パスロジックを決定する一実施例の構成図である。

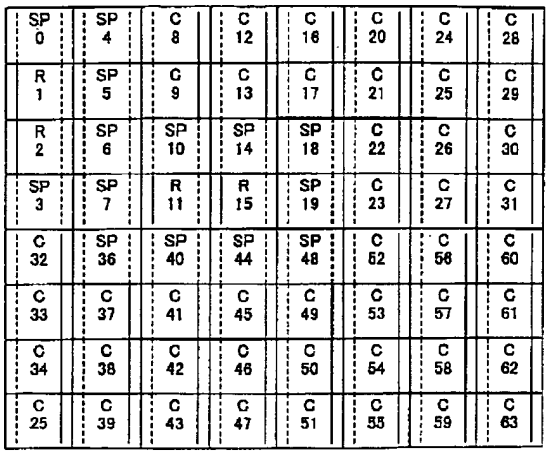
【符号の説明】

２９０１　画像データインタフェース
２９０２　ＤＣレベルシフタ
２９０３　ウェーブレット変換器
２９０５　スカラー量子化／逆量子化器
２９０６　プリコーダ
２９０７　パケットヘッダ処理器
２９０８　ビットモデリングＭＱ符号器
２９１１　符号メモリ
Ａ，Ｂ　ワークメモリ

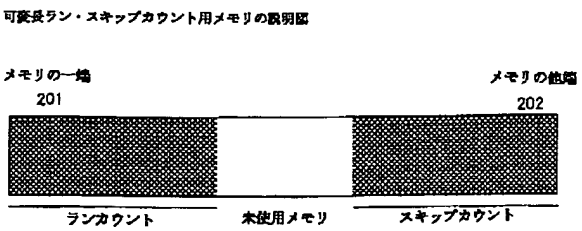
【図１】



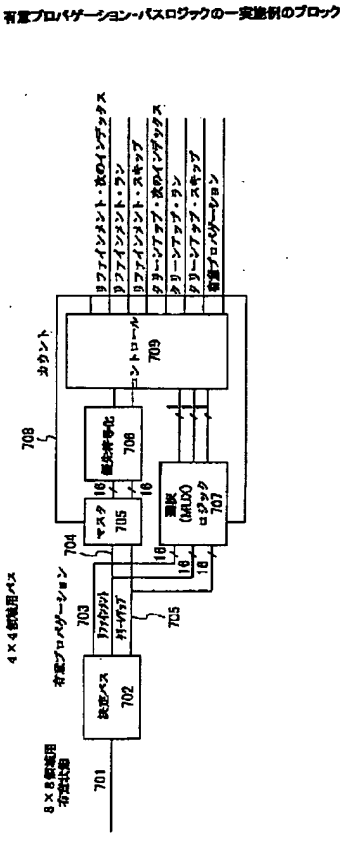
【図２Ａ】



【図２Ｂ】

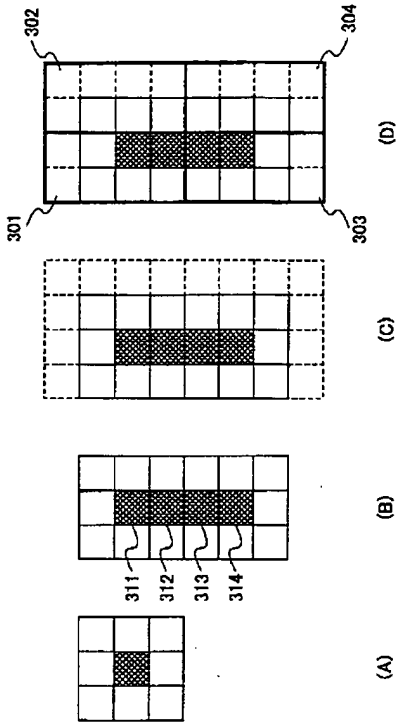


【図７】



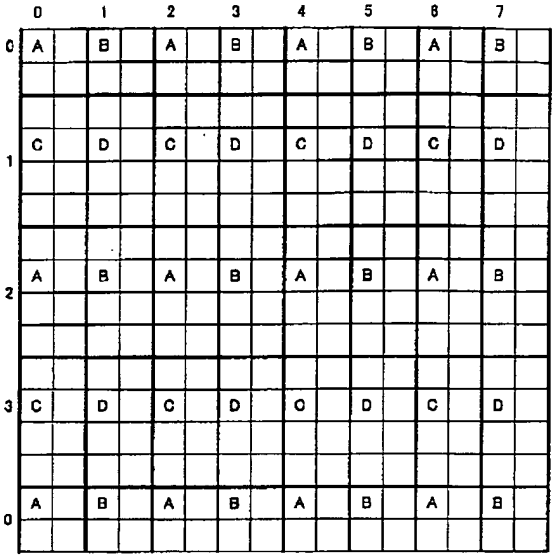
【図３】

近傍係数とメモリ構成の説明図



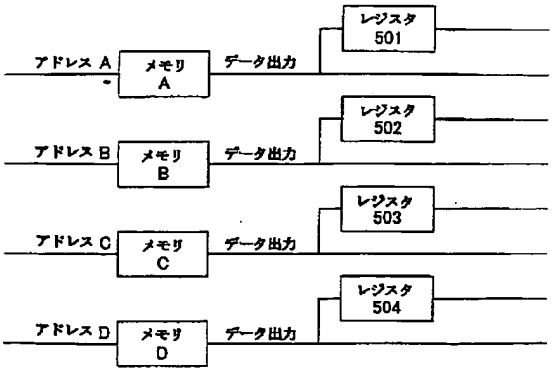
【図４】

有意メモリ構成の一実施例の説明図



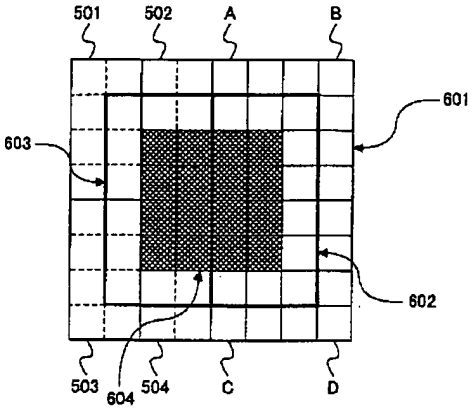
【図５】

有意プロパゲーション・パスに使用されるメモリ及びレジスタの説明図



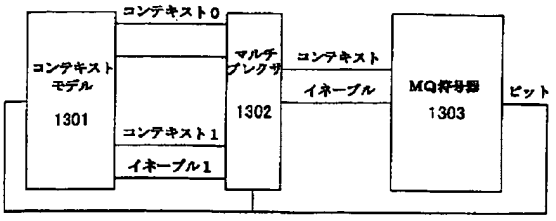
【図６】

有意状態の説明図



【図１３】

2重コンテキスト生成ロジックの一実施例のブロック図



【図８】

コンテキストモデルの一実施例の性能の説明図

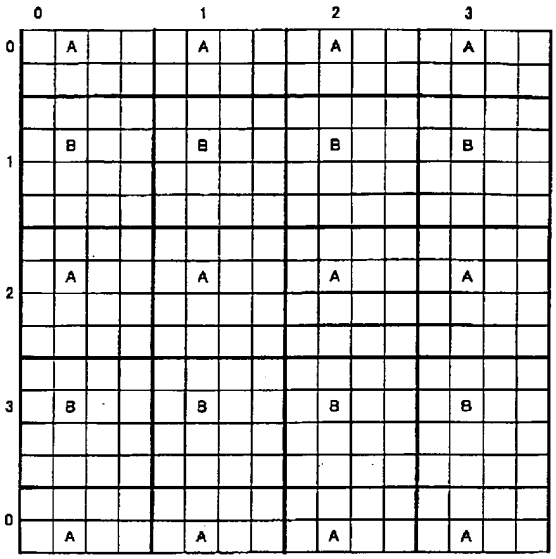
SPパス　：SPパス保護用の１０＊２
　　CとRパス保護用の８
Cパス　：Cパス保護用の４×２
Rパス　：Rパス用の２
合計：３８クロック＝１保護あたり２．２５クロック

SP=有意プロパゲーション
C=クリーンアップ
R=リファインメント

SP	C	C	SP	SP
SP	C	C	SP	R
SP	C	C	SP	R
SP	C	C	SP	SP

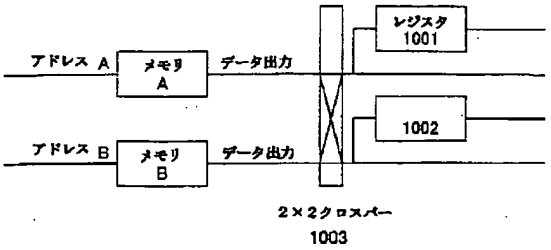
【図９】

有意メモリの構成の一実施例の説明図



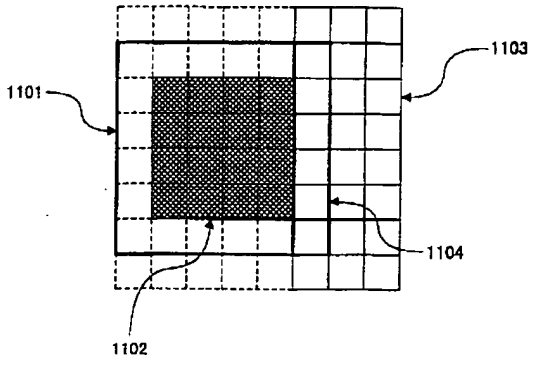
【図１０】

有意プロパゲーション・パスに使用されるメモリ及びレジスタの説明図



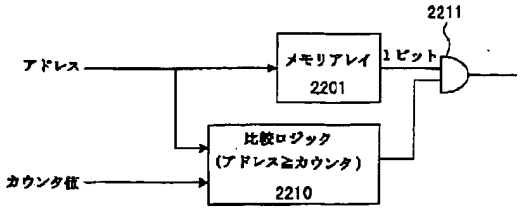
【図１１】

メモリ及びレジスタの使用法の説明図



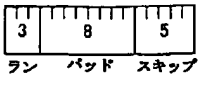
【図２２Ｂ】

制御ロジックの一実施例のブロック図



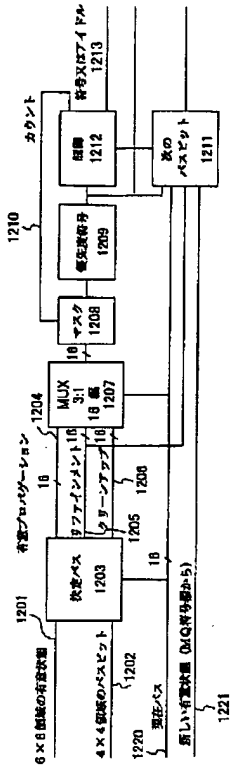
【図３０】

16ビット語の一例の説明図



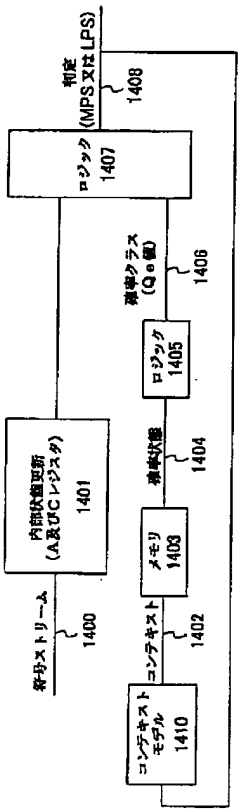
【図１２】

バス決定ロジックの一実施例のブロック図



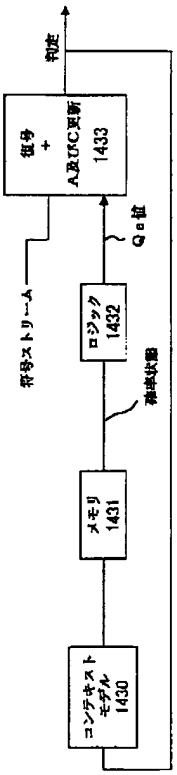
【図１４Ａ】

初期コンテキスト型MQ符号器の一実施例のブロック図



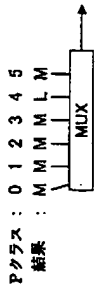
【図１４Ｂ】

典型的な復号化実現方式の一実施例の説明図



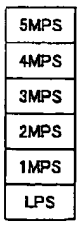
【図１６Ｂ】

マルチプレクサのブロック図

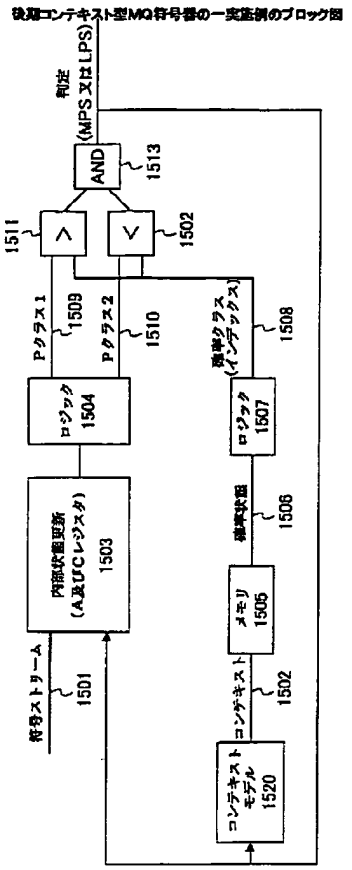


【図１７】

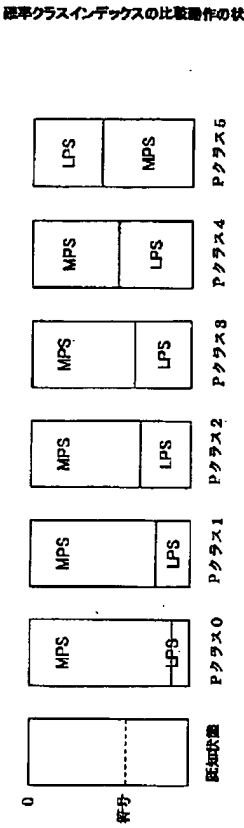
多重MPS復号化のためのインターバルの説明図



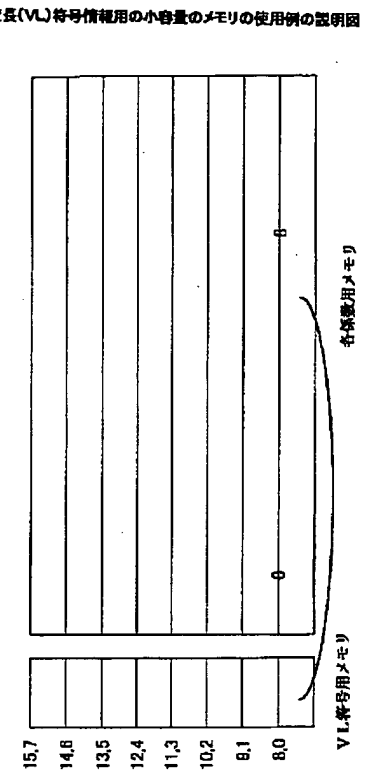
【図１５】



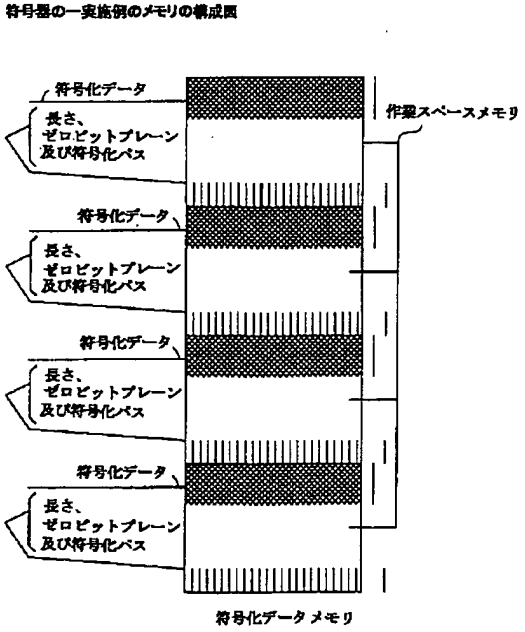
【図１６Ａ】



【図２３】

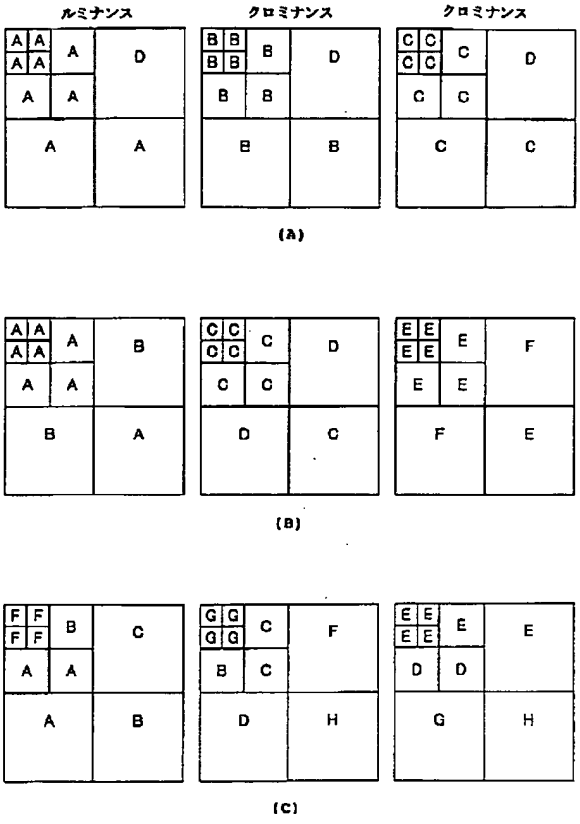


【図２１】



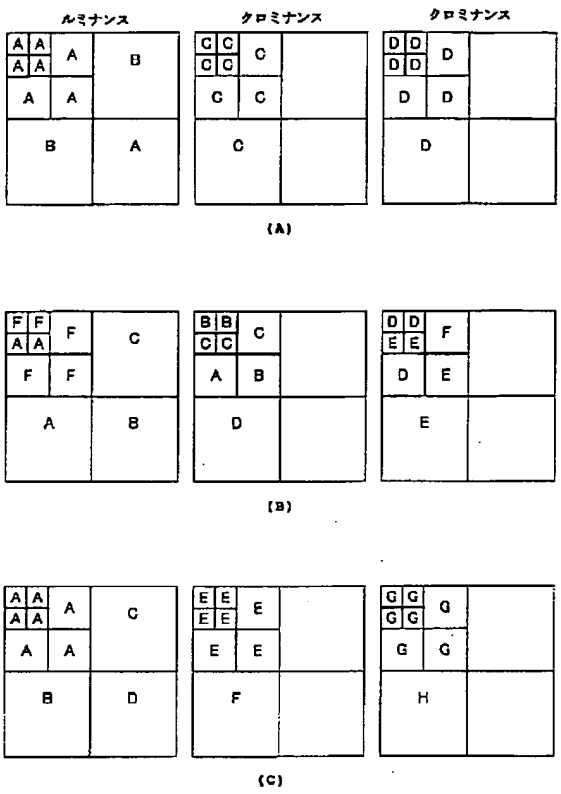
【図１８】

並列的な符号ブロックの割当の一実施例の説明図



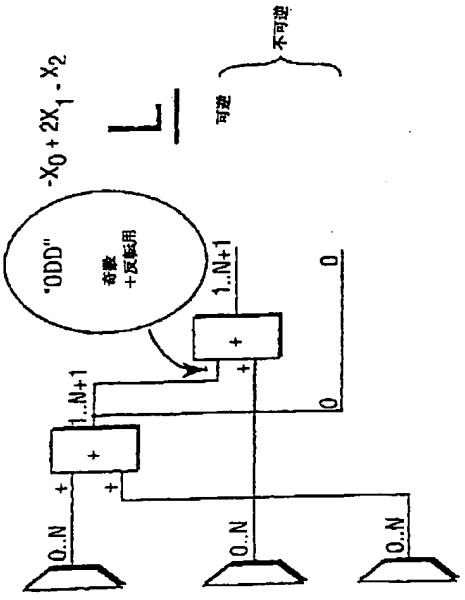
【図１９】

並列的な符号ブロックの割当の一実施例の説明図



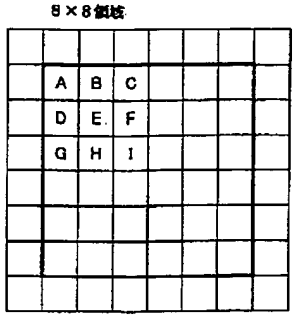
【図２５Ｂ】

ハイパスフィルタの一実施例のブロック図



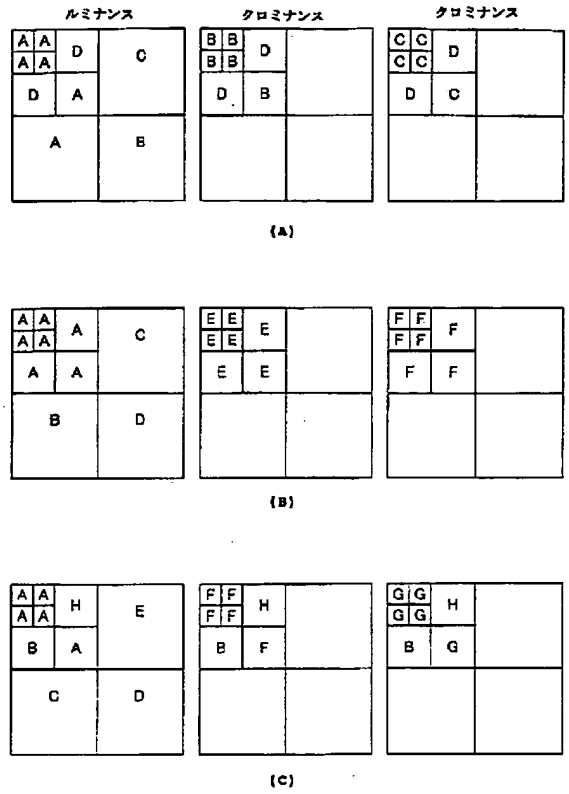
【図３１】

符号化パスを決定する有意状態ビットの8×8領域の説明図



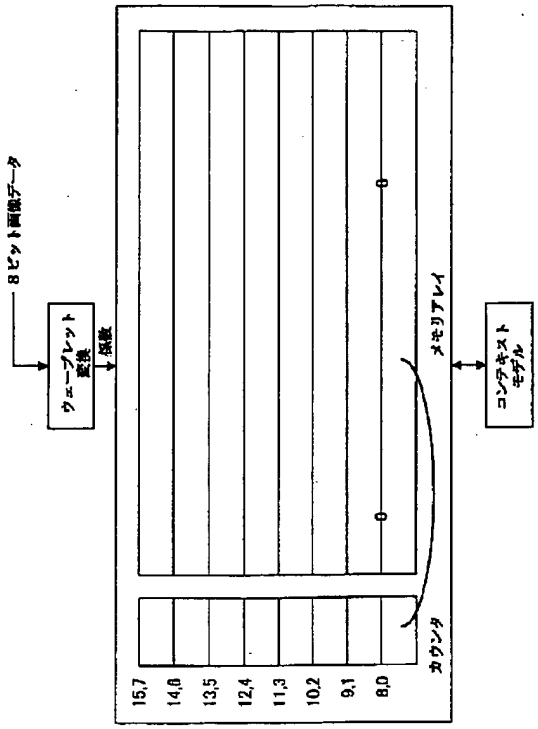
【図２０】

並列的な符号ブロックの割当の他の一実施例の説明図



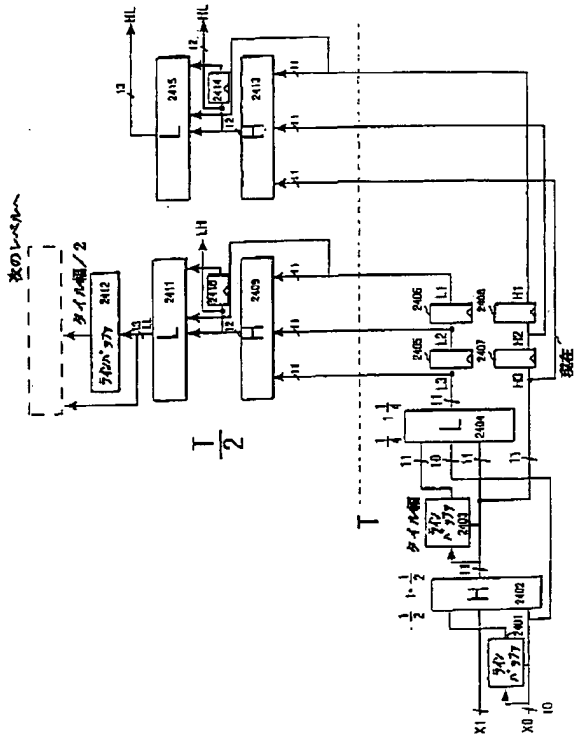
【図２２Ａ】

メモリのビットプレーンを使用する例の説明図



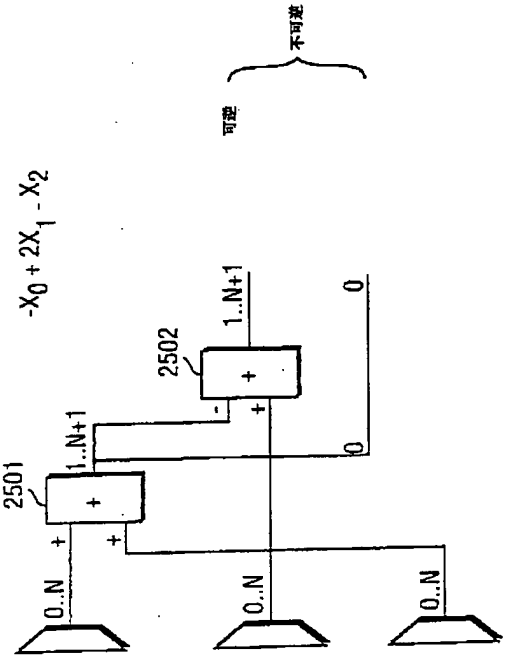
【図２４】

順変換の一実施例のブロック図



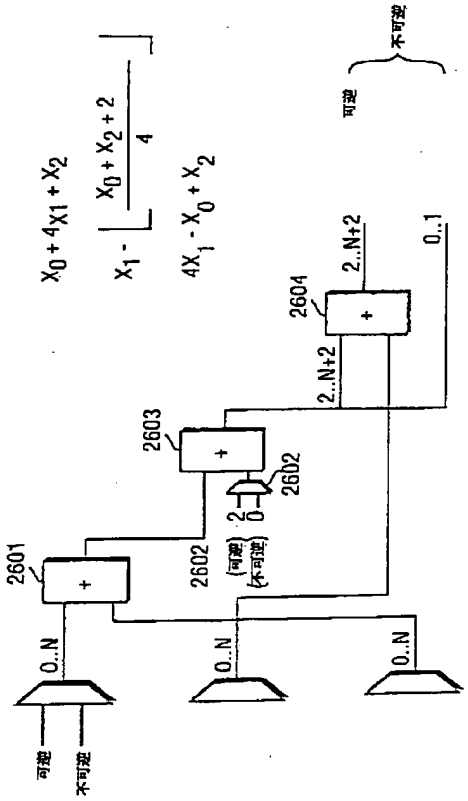
【図２５Ａ】

ローパスフィルタの一実施例のブロック図



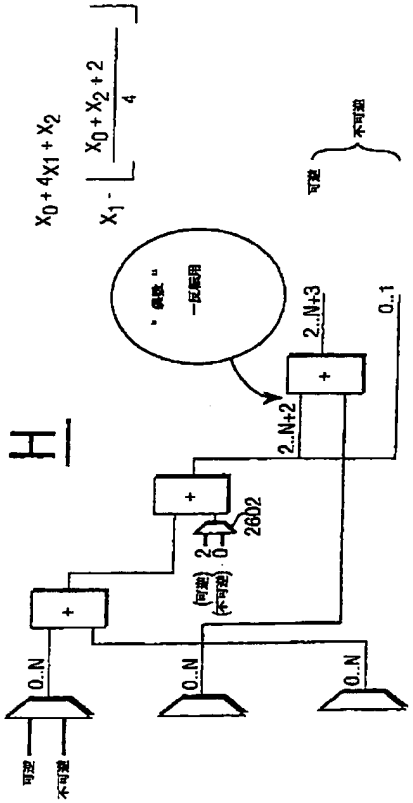
【図２６Ａ】

ハイパスフィルタの一実施例のブロック図



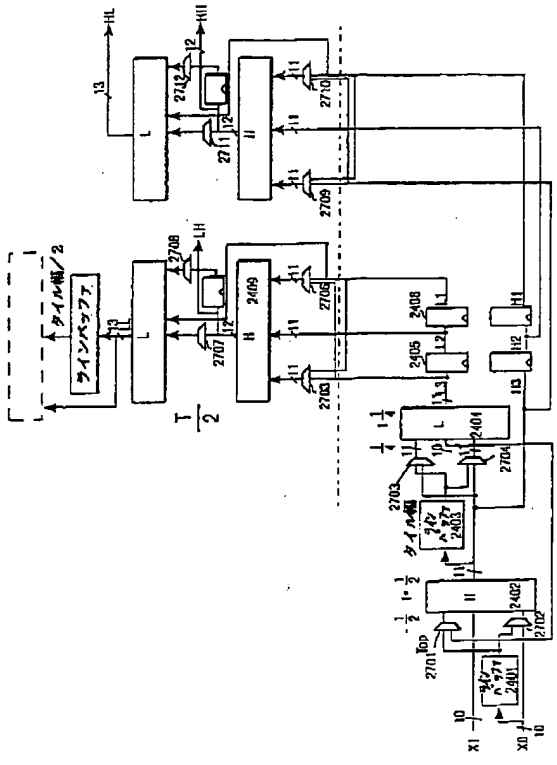
【図２６Ｂ】

ローパスフィルタの一実施例のブロック図



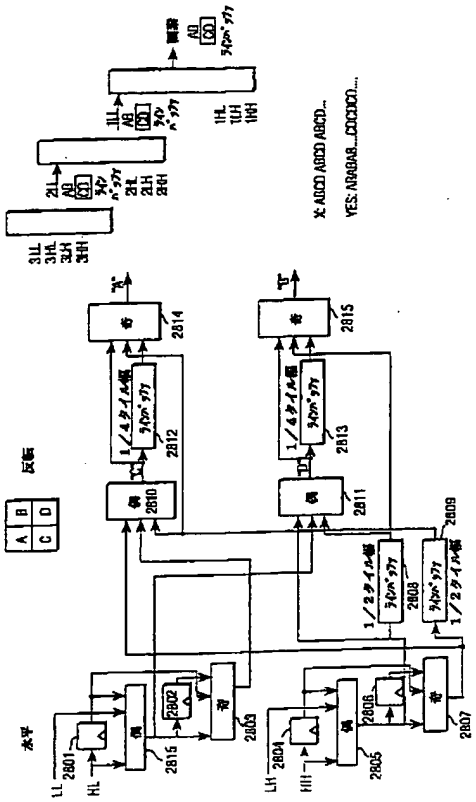
【図２７】

順変換の他の一実施例のブロック図

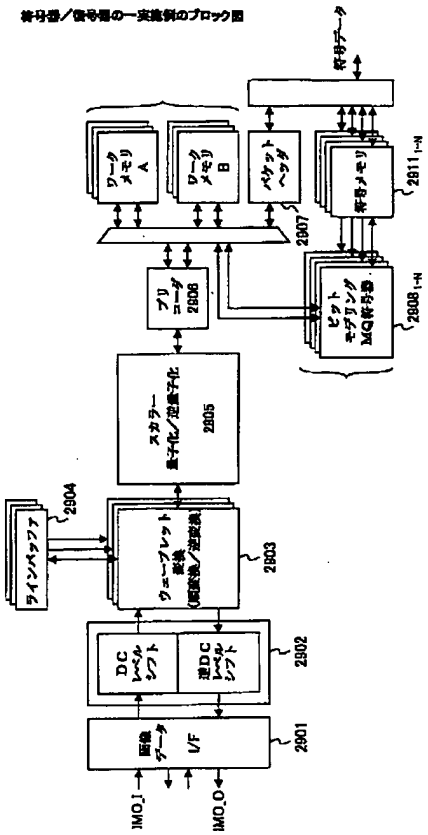


【図２８】

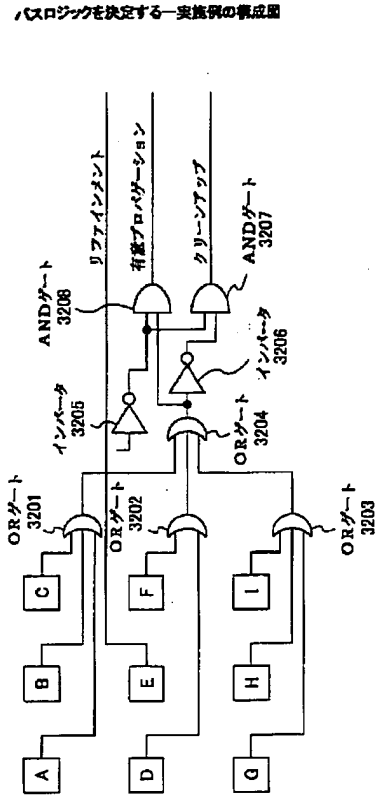
逆変換の一実施例のブロック図



【図２９】



【図３２】



【手続補正書】
【提出日】平成１４年5月8日（２００２．５．８）
【手続補正１】
【補正対象書類名】明細書
【補正対象項目名】特許請求の範囲
【補正方法】変更
【補正内容】
【特許請求の範囲】
【請求項１】　係数のグループに関して有意プロパゲーション・パスを実行する手順と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する手順と、
リファインメント・サブビットプレーン・パスを実行する手順と、を有し、
リファインメント・サブビットプレーン・パスは、
リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスする手順と、
リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手順と、
メモリから取得したリファインメント・ビットを符号化する手順と、によって実行される、方法。
【請求項２】　係数のグループに関して有意プロパゲーション・パスを実行する手段と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する手段と、
リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスし、
リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスし、
メモリから取得したリファインメント・ビットを符号化することにより、リファインメント・サブビットプレーン・パスを実行する手段と、を有する装置。
【請求項３】　係数のグループに関して有意プロパゲーション・パスを実行する機能と、

後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する機能と、
リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスし、リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスし、メモリから取得したリファインメント・ビットを符号化することによって、リファインメント・サブビットプレーン・パスを実行する機能と、をコンピュータに実現させるためのプログラム。

【請求項４】 係数のグループに関して有意プロパゲーション・パスを実行する手順と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する手順と、
クリーンアップ・サブビットプレーン・パスを実行する手順と、を有し、
クリーンアップ・サブビットプレーン・パスは、
クリーンアップ・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスする手順と、
クリーンアップ・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手順と、
メモリから取得したクリーンアップ・ビットを符号化する手順と、によって実行される、方法。

【請求項５】 係数のグループに関して有意プロパゲーション・パスを実行する手段と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する手段と、
クリーンアップ・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスし、
クリーンアップ・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスし、
メモリから取得したクリーンアップ・ビットを符号化することにより、クリーンアップ・サブビットプレーン・パスを実行する手段と、を有する装置。

【請求項６】 係数のグループに関して有意プロパゲーション・パスを実行する機能と、
後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造を作成する機能と、
クリーンアップ・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するためデータ構造にアクセスし、クリーンアップ・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスし、メモリから取得したクリーンアップ・ビットを符号化することによって、クリーンアップ・サブビットプレーン・パスを実行する機能と、をコンピュータに実現させるためのプログラム。

【請求項７】 リファインメント・ビットのランのサイズを指定するリファインメント・ビットランカウント、リファインメント・ビットのランの間でスキップすべき係数の個数を指定するリファインメント・ビットスキップカウント、クリーンアップ・ビットのランのサイズを指定するクリーンアップ・ビットランカウント、及び、クリーンアップ・ビットのランの間でスキップすべき係数の個数を指定するクリーンアップ・ビットスキップカウントを収容し、リファインメント・パス又はクリーンアップ・パスの間に処理されるべき係数のグループ内での係数の場所を指示する、少なくとも一つのデータ構造を、コンテキストモデルが読み出す手順と、
コンテキストモデルが、少なくとも一つのデータ構造内の情報に基づいてメモリにアクセスする手順と、
を有する方法。

【請求項８】 メモリと、メモリに接続されたコンテキストモデルとを具備し、
少なくとも一つのデータ構造は、リファインメント・ビットのランのサイズを指定するリファインメント・ビットランカウント、リファインメント・ビットのランの間でスキップすべき係数の個数を指定するリファインメント・ビットスキップカウント、クリーンアップ・ビットのランのサイズを指定するクリーンアップ・ビットランカウント、及び、クリーンアップ・ビットのランの間でスキップすべき係数の個数を指定するクリーンアップ・ビットスキップカウントを収容し、
コンテキストモデルは、リファインメント・パス又はクリーンアップ・パスの間に処理されるべき係数のグループ内での係数の場所を指示する少なくとも一つのデータ構造を読み出し、
コンテキストモデルは、少なくとも一つのデータ構造内の情報に基づいてメモリにアクセスする、復号器。

【請求項９】 リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するため、後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造にアクセスする手順と、
リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手順と、
メモリから取得したリファインメント・ビットを符号化する手順と、を有する方法。

【請求項１０】 リファインメント・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するため、後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造にアクセスする手段と、

リファインメント・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手段と、
メモリから取得したリファインメント・ビットを符号化する手段と、を有する装置。
【請求項１１】 クリーンアップ・サブビットプレーン・パスの処理の際にはスキップされるべき係数を識別する情報を獲得するため、後続のパスで処理されるべき係数のグループ内での係数の場所を示すデータ構造にアクセスする手順と、
クリーンアップ・パスに含まれるとして識別された係数だけにアクセスするため、獲得された情報を用いて、係数のグループを保持するメモリにアクセスする手順と、
メモリから取得したクリーンアップ・ビットを符号化する手順と、を有する方法。
【請求項１２】 ランカウントを保持する第１の部分、スキップカウントを保持する第２の部分、及び、第１の部分と第２の部分を分離する第３の部分を有するメモリと、
上記メモリに接続され、上記メモリから同時に獲得されたランカウント及びスキップカウントを復号化する復号化ハードウェアと、を具備した情報復号化装置。
【請求項１３】 有意プロパゲーション・パスを実行する方法であって、
領域に対する有意状態情報に応じて、処理される情報が有意プロパゲーション・パス、リファインメント・パス、又は、クリーンアップ・パスに属するかどうかを判定する手順と、
領域内のサブ領域が、有意プロパゲーション・パス、リファインメント・パス、又は、クリーンアップ・パスの係数を有するかどうかを表わす信号をアサートする手順と、
リファインメント・パス及びクリーンアップ・パスに対するランカウント及びスキップカウントを識別すると共に、有意プロパゲーション・パスの係数のビットを符号化する手順と、
リファインメント・パスにあるものとして認定された係数のビット及びクリーンアップ・パスにあるものとして認定された係数のビットを処理する手順と、を有する方法。
【請求項１４】 有意プロパゲーション・パスを実行する装置であって、
領域に対する有意状態情報に応じて、処理される情報が有意プロパゲーション・パス、リファインメント・パス、又は、クリーンアップ・パスに属するかどうかを判定する手段と、
領域内のサブ領域が、有意プロパゲーション・パス、リファインメント・パス、又は、クリーンアップ・パスの係数を有するかどうかを表わす信号をアサートする手段と、
リファインメント・パス及びクリーンアップ・パスに対するランカウント及びスキップカウントを識別すると共に、有意プロパゲーション・パスの係数のビットを符号化する手段と、
リファインメント・パスにあるものとして認定された係数のビット及びクリーンアップ・パスにあるものとして認定された係数のビットを処理する手段と、を有する装置。
【請求項１５】 有意プロパゲーション・パスを実行する装置であって、
第１の所定のサイズの第１の領域に対する有意状態情報を受け取る入力、並びに、第２の所定のサイズの第２の領域における各係数と関連したパスが有意プロパゲーション・パスであるか、リファインメント・パスであるか、又は、クリーンアップ・パスであるかを示す第１の出力標識の組を具備した決定パスユニットと、
出力標識に接続された入力、次の非リファインメント・ビット出力、次の非クリーンアップ・ビット出力、及び、現在のパス標識出力を具備した処理ユニットと、
処理ユニットからの出力へ接続され、上記処理ユニットからの出力に応じて、リファインメント・ビットデータ構造への次のインデックスの標識、リファインメント・ラン標識、リファインメント・スキップ標識、及び、有意プロパゲーション標識を生成する制御ユニットと、が設けられた装置。
【請求項１６】 領域に対する有意状態情報、領域の部分集合内の各係数に対するパスビット、及び、現在パス標識を受けるため接続された入力を具備し、領域の部分集合内の各係数に対するパスを示すパス標識を生成する判定パスロジックと、
判定パスロジックから出力された信号及び現在パス標識を受ける入力を具備し、有意プロパゲーション・パス、リファインメント・パス及びクリーンアップ・パスのうちの一つのパスと関連付けられた出力標識を生成する選択ロジックと、選択ロジックに接続され、選択ロジックからの出力標識に応じて現在パスにおける次の係数を示す次の係数ロジックと、
一つのパスにおける次の係数を示す出力を受けるよう接続され、有意プロパゲーション・パス、リファインメント・パス及びクリーンアップ・パスにおける係数を符号化し、符号化される係数の中には次の係数ロジックによって示された有意プロパゲーション・パス、リファインメント・パス及びクリーンアップ・パスのうちのいずれかにおける係数が含まれている、制御ロジックと、を有する装置。
【請求項１７】 領域に対する有意状態情報、領域の部分集合内の各係数に対するパスビット、及び、現在パス標識を受けるため接続された入力を具備し、領域の部分集合内の各係数に対するパスを示すパス標識を生成する

判定パスロジックと、
判定パスロジックから出力された信号及び現在パス標識を受ける入力を具備し、有意プロパゲーション・パス、リファインメント・パス及びクリーンアップ・パスのうちの一つのパスと関連付けられた出力標識を生成する選択ロジックと、
選択ロジックからの出力標識及び処理中の領域の部分集合内の現在の係数の識別標識を受けるよう接続された入力を具備し、マスクされた出力標識を出力するマスクユニットと、
マスクユニットに接続され、現在のパスにおける次の係数を示す出力を具備し、現在のパスにおける各係数の場所を見つける優先度符号器と、
一つのパスにおける次の係数を示す出力を受けるよう接続され、有意プロパゲーション・パス、リファインメント・パス及びクリーンアップ・パスにおける係数を符号化し、符号化される係数の中には次の係数ロジックによって示された有意プロパゲーション・パス、リファインメント・パス及びクリーンアップ・パスのうちのいずれかにおける係数が含まれている、制御ロジックと、
次のパスビットの状態を示すため次のパスビット標識を設定する次のパスビットロジックと、を有する装置。

【請求項１８】 復号化されている現在のビットが０であるときのコンテキストと、復号化されている現在のビットが、係数が有意になったことを示す１であるときのコンテキストの二つのコンテキストを生成する手順と、
現在のビットに続くビットの復号化に使用するため、既に復号化された現在のビットに基づいて、二つのコンテキストのうちの一方を選択する手順と、
選択された一方のコンテキストを使用して現在のビットに続くビットを復号化する手順と、を有する方法。

【請求項１９】 復号化されている現在のビットが０であるときのコンテキストと、復号化されている現在のビットが、係数が有意になったことを示す１であるときのコンテキストの二つのコンテキストを生成するコンテキストモデルと、
コンテキストモデルに接続され、選択信号に応じて二つのコンテキストのうちの一方を出力する選択器と、
選択器に接続され、選択器から一方のコンテキストを受ける符号器と、を有する復号器。

【請求項２０】 復号化されている現在のビットが０であるときのコンテキストと、復号化されている現在のビットが、係数が有意になったことを示す１であるときのコンテキストの二つのコンテキストを生成する機能と、
現在のビットに続くビットの復号化に使用するため、既に復号化された現在のビットに基づいて、二つのコンテキストのうちの一方を選択する機能と、
選択された一方のコンテキストを使用して現在のビットに続くビットを復号化する機能と、をコンピュータに実現させるためのプログラム。

【請求項２１】 ＭＱ符号器とＡレジスタ及びＣレジスタとを用いて情報を復号化する復号器であって、
内部状態を更新する更新ロジックと、
コンテキストに応じて現在の確率クラスインデックスを生成する第１のロジックと、が設けられ、
更新ロジックによる内部状態の更新は、コンテキストの生成と現在の確率クラスインデックスの生成の少なくとも一方と重なり、
符号ストリームに応じて、Ａレジスタ及びＣレジスタによって指定されたインターバルと関連付けられた確率クラスインデックスのペアを生成する確率クラス生成ロジックと、
第１のロジック及び確率クラス生成ロジックと接続され、現在の確率クラスインデックスに対応する確率クラスが確率クラスインデックスのペアに対応した確率クラスの範囲内に含まれるかどうかを判定する比較ロジックと、
比較ロジックに接続され、現在の確率クラスインデックスと確率クラスインデックスのペアとの間の比較の結果に基づく判定結果を出力する出力ロジックと、が更に設けられている復号器。

【請求項２２】 ＭＱ符号器とＡレジスタ及びＣレジスタとを用いて情報を復号化する復号器であって、
内部状態を更新する更新ロジックと、
コンテキストに応じて現在の確率クラスインデックスを生成する第１のロジックと、が設けられ、
更新ロジックによる内部状態の更新は、コンテキストの生成と現在の確率クラスインデックスの生成の少なくとも一方と重なり、
コンテキストに応じて確率状態を出力するメモリと、
メモリに接続され、確率状態に応じて現在の確率クラスインデックスを生成する第２のロジックと、
符号ストリームに応じて、Ａレジスタ及びＣレジスタによって指定されたインターバルと関連付けられた確率クラスインデックスのペアを生成する確率クラス生成ロジックと、
第１のロジック及び確率クラス生成ロジックと接続され、比較器のペアを含み、現在の確率クラスインデックスが確率クラスインデックスのペアの範囲内に含まれるかどうかを判定する比較ロジックと、
比較ロジックに接続され、現在の確率クラスインデックスと確率クラスインデックスのペアとの間の比較の結果に基づく判定結果を出力する出力ロジックと、が更に設けられ、
出力ロジックは比較器のペアの出力に接続されたＡＮＤゲートを具備し、
出力ロジックのＡＮＤゲートは、現在の確率クラスインデックスに対応した確率クラスが確率クラスインデックスのペアによって決まる確率クラスの範囲に含まれない場合に、高確率シンボルを出力する、復号器。

【請求項２３】 Ａレジスタ及びＣレジスタを含むＭＱ符号器を用いて符号ストリームを復号化する復号化システムで使用するための復号化方法であって、
Ａレジスタ及びＣレジスタによって決まるインターバルに対する符号ストリームの位置を判定する手順と、
Ａレジスタ及びＣレジスタによって決まるインターバルに対する符号ストリームの位置と確率シンボルの推定値であるＱｅ値の差が２以上である場合に、多数のビットを復号化する手順と、を有する復号化方法。
【請求項２４】 Ａレジスタ及びＣレジスタを含むＭＱ符号器を用いて符号ストリームを復号化する復号化システムで使用するための復号化装置であって、
Ａレジスタ及びＣレジスタによって決まるインターバルに対する符号ストリームの位置を判定する手段と、
Ａレジスタ及びＣレジスタによって決まるインターバルに対する符号ストリームの位置と確率シンボルの推定値であるＱｅ値の差が２以上である場合に、多数のビットを復号化する手段と、を有する復号化装置。
【請求項２５】 Ａレジスタ及びＣレジスタによって決まるインターバルに対する符号ストリームの位置を判定する機能と、
Ａレジスタ及びＣレジスタによって決まるインターバルに対する符号ストリームの位置と確率シンボルの推定値であるＱｅ値の差が２以上である場合に、多数のビットを復号化する機能と、をコンピュータに実現させるためのプログラム。
【請求項２６】 第１のラインバッファと、
第１の入力係数、第２の入力係数、及び、第１のラインバッファの出力を受ける入力、並びに、第１の出力を具備した第１のハイパスフィルタと、
第１の出力を受けるように第１のハイパスフィルタに接続され、第２のラインバッファ出力を具備した第２のラインバッファと、
第１の入力係数、第１の出力、及び、第２のラインバッファ出力を受ける入力、並びに、第２の出力を具備した第１のローパスフィルタと、
第２の出力に接続され、第１の遅延出力及び第２の遅延出力を具備した第２の縦続接続された遅延器のペアと、
第２の出力、第１の遅延出力、及び、第２の遅延出力に接続された入力を具備し、ＬＨサブバンドにおける係数である第３の出力を生成する第２のハイパスフィルタと、
第３の出力に接続され、第３の遅延出力を具備した第１の遅延器と、
第３の出力、第３の遅延出力、及び、ＬＬサブバンドにおける係数に対応した第４の出力に接続された入力を具備し、第４の出力を生成する第２のローパスフィルタと、
第１の出力に接続され、第４の遅延出力及び第５の遅延出力を具備した第２の縦続接続された遅延器のペアと、
第１の出力、第４の遅延出力、及び、第５の遅延出力に接続された入力を具備し、ＨＨサブバンドにおける係数である第５の出力を生成する第３のハイパスフィルタと、
第５の出力に接続され、第６の遅延出力を具備した第２の遅延器と、
第５の出力、第６の遅延出力、及び、第５の遅延出力に接続された入力を具備し、ＨＬサブバンドにおける係数に対応した第６の出力を生成する第３のローパスフィルタと、を有するウェーブレット変換フィルタ。
【請求項２７】 入力データを複数の符号ブロックに分割する手順と、
複数のＭＱ符号器の中の各ＭＱ符号器によって行なわれる符号化の量ができる限り均衡するように並列的に複数の符号ブロックを符号化するため、複数の符号ブロックを、符号ブロック単位で、複数のＭＱ符号器に割り当てる手順と、を有する方法。
【請求項２８】 入力データを複数の符号ブロックに分割する手順と、
複数のＭＱ符号器の中の各ＭＱ符号器によって行なわれる符号化の量ができる限り均衡するように並列的に複数の符号ブロックを符号化するため、複数の符号ブロックを、符号ブロック単位で、複数のＭＱ符号器に割り当てる手順と、を有する装置。
【請求項２９】 各係数のビット長がＮビットよりも長い複数の係数を生成するため、１回以上のウェーブレット変換を適用する手順と、
各係数のＮ個のビットプレーンを、複数の行を含む第１のメモリに格納する手順と、を有し、
第１のメモリの各行の各記憶場所は、複数の係数の比較的有意なビットプレーンと複数の係数のあまり有意ではないビットプレーンのいずれか一方のビットを格納し、
Ｎ個のビットプレーンを第１のメモリへ格納する手順は、比較的有意なビットプレーンのビットが格納され始め、あまり有意ではないビットプレーンの複数の係数のビットが格納されなくなる第１のメモリの各行内での場所を示すため、第１のメモリの各行に対する標識を格納する手順を含む、係数符号化方法。
【請求項３０】 各係数のビット長がＮビットよりも長い複数の係数を生成するため、１回以上のウェーブレット変換を適用する手段と、
各係数のＮ個のビットプレーンを、複数の行を含む第１のメモリに格納する手段と、を有し、
第１のメモリの各行の各記憶場所は、複数の係数の比較的有意なビットプレーンと複数の係数のあまり有意ではないビットプレーンのいずれか一方のビットを格納し、
Ｎ個のビットプレーンを第１のメモリへ格納する手段は、比較的有意なビットプレーンのビットが格納され始め、あまり有意ではないビットプレーンの複数の係数のビットが格納されなくなる第１のメモリの各行内での場

所を示すため、第１のメモリの各行に対する標識を格納する手段を含む、係数符号化装置。

【手続補正２】
【補正対象書類名】明細書
＊【補正対象項目名】００８２
【補正方法】変更
【補正内容】
＊【００８２】

```
count = 0
while (count < 16)
    mask = (1 << count)-1
    refinement_masked = refinement | mask
    use priority encoder to find next non-refinement bit
    cleanup_mask = clean_up | mask
    use priority encoder to fine next non-cleanup bit
    if current bit is in significance propagation pass
        process coefficient as significance propagation
        count = count + 1
    else if current bit in refinement pass
        N = "next non-refinement bit" − count
        process N bits as refinement pass
        count = count + N
    else
        N = "next non-cleanup bit" − count
        process N bits as cleanup pass
        count = count + N                /* 。
```

※～７は、カウンタがメモリの最後に値を格納した場合にメモリに収容するか、又は、カウンタはメモリ内のビットプレーン８～１５に対する開始アドレスを指定し、対応したビットプレーン０～７を丸め処理（量子化）する必要がある。かくして、カウントは、サイドバンド情報として作用し、行の始まりからカウントによって指定された行内の位置までメモリアレイに保持された情報が必要の無いデータになったことを示す。

【手続補正３】
【補正対象書類名】明細書
【補正対象項目名】０１５３
【補正方法】変更
【補正内容】
【０１５３】符号化中に、係数は、零ビットプレーンの数が既知になる前に保存される。カウンタは、上位のビットプレーン８～１５に対する初期零の数をカウントする。ビットプレーン８～１５が全て零である限り、メモリは、対応したビットプレーン０～７に対する情報（大きさ）を保持する。ビットプレーン８～１５に１が出現した場合、対応したカウンタは停止し、メモリは対応したビットプレーン８～１５の情報を保持する。符号ブロックの符号化の最後に、カウンタがビットプレーン８～１５に対し全て零を指定し、対応したビットプレーン０※

```
write "1"
for each subband
    write "1"
    first_flag = 1
    for each code-block
        if not included then
            write "0"
        else
            write "1"
            if first_flag then
                write MZP in tag tree format
                first_flag = 0
            write zero bitplanes − MZP in tag tree format
```

【手続補正４】
【補正対象書類名】明細書
【補正対象項目名】０１６８
【補正方法】変更
【補正内容】
【０１６８】

```
                write coding passes
                determine minimum Lblock value
                write Lblock
                write length
```

ここで、Lblockは、JPEG 2000標準のセクションB.10.7.1に規定されている。

【手続補正５】
【補正対象書類名】明細書

```
            write "1"
            for each subband
                if layer 0 then write "1"
                for each code-block
                    if not included then
                        write "0"
                    else
                        write "1"
                        if code-block not already included then
                            if first_flag then
                                write MZP in tag tree format
                                first_flag = 0
                            write zero bitplanes − MZP in tag tree format
                            set already included
                        write coding passes
                        determine minimum Lblock value
                        write Lblock
                        write length
```

"already included"情報は、各符号ブロックに対し別個のビットでも構わない。或いは、零ビットプレーンの使用されない値を"already included"を指定するため使用してもよい。たとえば、１４個のビットプレーンが存在※する場合、零ビットプレーンを１５(0xF)にセットすることによって、"already included"を示すことが可能である。

＊【補正対象項目名】０１７３
【補正方法】変更
【補正内容】
【０１７３】

フロントページの続き

(72)発明者 佐藤 豊
アメリカ合衆国，カリフォルニア 94025，メンロ・パーク，サンド・ヒル・ロード 2882番，スイート 115 リコーイノベーション内

Fターム(参考) 5C059 MA00 MA24 MA35 MC11 ME01 ME05 ME06 RB02 RB17 UA02 UA05 UA12 UA14
5C078 AA04 BA53 CA01 DA01 DA02 DA11 DA12
5J064 AA01 BA08 BA09 BA16 BB13 BC01 BC08 BC16 BC23 BC29 BD02 BD03 BD06

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2003-008445

(43)Date of publication of application : 10.01.2003

---

(51)Int.Cl. H03M 7/30
H03M 7/40
H03M 7/46
H04N 1/41
H04N 7/30

---

(21)Application number : 2002-095109 (71)Applicant : RICOH CO LTD

(22)Date of filing : 29.03.2002 (72)Inventor : SCHWARTZ EDWARD L
SATO YUTAKA

---

(30)Priority
Priority number : 2001 823733
Priority date : 30.03.2001
Priority country : US

---

(54) CONTEXT MODEL ACCESS TO MEMORY BASED ON RUN AND SKIP COUNTS

(57)Abstract:

PROBLEM TO BE SOLVED: To provide a method and apparatus for coding and decoding information.

SOLUTION: In one embodiment, the apparatus comprises a memory and decoding hardware. The memory stores run counts and/or skip counts and the decoding hardware decodes a run count and/or a skip count obtained from the memory during decoding.

LEGAL STATUS [Date of request for examination] 03.02.2005
[Date of sending the examiner's decision of rejection]
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

1.This document has been translated by computer. So the translation may not reflect the original precisely.

2.**** shows the word which can not be translated.

3.In the drawings, any words are not translated.

CLAIMS

[Claim(s)]

[Claim 1] The procedure of performing significant propagation pass about the group of a multiplier, The procedure which creates the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, It has the procedure of performing refinement MENTO subbit plane pass. Refinement MENTO subbit plane pass The procedure which accesses DS in order to acquire the information which identifies the multiplier which should be skipped in the case of processing of refinement MENTO subbit plane pass, The procedure which accesses the memory which holds the group of a multiplier using the acquired information in order to access only the multiplier identified noting that it was contained in refinement MENTO pass, the procedure which encodes the refinement MENTO

bit acquired from memory, and the approach performed as be alike.

[Claim 2] It is the approach according to claim 1 by which two or more refinement MENTO bits are acquired, and a non-refining MENTO bit is disregarded.

[Claim 3] The group of a multiplier is the approach according to claim 1 of constituting a sign block.

[Claim 4] The procedure which creates the above-mentioned DS is the approach according to claim 1 of creating at least one DS by creating the DS which describes the location of the multiplier of refinement MENTO subbit plane pass to the group of a multiplier.

[Claim 5] It is the approach according to claim 4 by which the 1st DS is created during significant propagation pass for a refinement MENTO bit, and the 2nd DS is created for a clean-up bit.

[Claim 6] DS is an approach containing the indicator of each run of a refinement MENTO bit, and the number of the multiplier which should be skipped within the group of a multiplier before the run next to a refinement MENTO bit according to claim 4.

[Claim 7] At least for one of the indicator of each run of a refinement MENTO bit, and the numbers of the multiplier which should be skipped, a count 1 is [a

symbolic language 000 and a count 2] an approach according to claim 6 by which a symbolic language 001 and a count 3 are expressed by the symbolic language 010, and a count 4 is expressed by a symbolic language 011 and the count 5 with a variable-length sign [like a symbolic language 1111111111011] whose count 4096 is like [a symbolic language 1 trillion and a count 6] a symbolic language billion and the following.

[Claim 8] At least one of the indicator of each run of a refinement MENTO bit and the numbers of the multiplier which should be skipped a count 1 -- a symbolic language 0 and a count 2 -- a symbolic language 01 and a count 3 -- a symbolic language 1100 and a count 4 -- a symbolic language 1101_0 and a count 5 -- a symbolic language 1101_1 and a count 6 -- a symbolic language 1110_0000 and a count 7 -- a symbolic language 1110_0001 -- similarly A count 21 a symbolic language 1110_1111 and a count 22 a symbolic language 1111_0000_0000_0000 and a count 23 like a symbolic language 1111_0000_0000_0001 and the following A count 4096 is an approach according to claim 6 expressed with a variable-length sign like a symbolic language 1111_1111_1110_1010.

[Claim 9] At least one of the indicator of each run of a refinement MENTO bit and

the numbers of the multiplier which should be skipped A count 1 the symbolic language 0 of gamma1 format, the symbolic language 0 of gamma format, and a count 2 The symbolic language 10_0 of gamma1 format, the symbolic language 100 of gamma format, and a count 3 The symbolic language 10_1 of gamma1 format, the symbolic language 110 of gamma format, and a count 4 The symbolic language 110_00 of gamma1 format, the symbolic language 10100 of gamma format, and a count 5 The symbolic language 110_01 of gamma1 format, the symbolic language 10110 of gamma format, and a count 6 The symbolic language 110_10 of gamma1 format, the symbolic language 11100 of gamma format, and a count 7 The symbolic language 110_11 of gamma1 format, the symbolic language 11110 of gamma format, and a count 8 The symbolic language 1110_000 of gamma1 format, the symbolic language 1010100 of gamma format, and a count 9 Like the symbolic language 1110_001 of gamma1 format, the symbolic language 1010110 of gamma format, and the following, a count 15 The symbolic language 1110_111 of gamma1 format, the symbolic language 1111110 of gamma format, and a count 16 The symbolic language 11110_0000 of gamma1 format, the symbolic language 101010100 of gamma format, and a count 32 The symbolic language 111110_00000 of gamma1

format, the symbolic language 10101010100 of gamma format, and a count 64 The symbolic language 111110_000000 of gamma1 format, the symbolic language 1010101010100 of gamma format, and a count 128 The symbolic language 11111110_0000000 of gamma1 format, the symbolic language 101010101010100 of gamma format, and a count 256 The symbolic language 111111110_00000000 of gamma1 format, the symbolic language 10101010101010100 of gamma format, and a count 512 The symbolic language 1111111110_000000000 of gamma1 format, the symbolic language 1010101010101010100 of gamma format, and a count 1024 Symbolic-language 1010101010101010101 00 of the symbolic language 11111111110_000000000 of gamma1 format and gamma format and a count 2048 Symbolic-language 1010101010101010101 0100 of the symbolic language 111111111110_0000000000 of gamma1 format and gamma format and a count 4096 symbolic-language 10101010101010101010 010100 of the symbolic language 1111111111110_00000000000 of gamma1 format, and gamma format, and ** -- the approach according to claim 6 expressed with a gamma sign [like].

[Claim 10] The indicator of each run of a refinement MENTO bit is an approach according to claim 1 held as an integer.

[Claim 11] An integer is an approach according to claim 10 expressed with the integer of the minimum size.

[Claim 12] The number of the multiplier which should be skipped is an approach according to claim 1 held as an integer.

[Claim 13] An integer is an approach according to claim 12 expressed with the integer of the minimum size.

[Claim 14] The indicator of each run of a refinement MENTO bit and the number of a multiplier which should be skipped are an approach according to claim 1 held as an integer.

[Claim 15] An integer is an approach according to claim 14 expressed with the integer of the minimum size.

[Claim 16] A means to perform significant propagation pass about the group of a multiplier, A means to create the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, It has a means to perform refinement MENTO subbit plane pass. Refinement MENTO subbit plane pass A means to access DS in order to acquire the information which identifies the multiplier which should be skipped in the case of processing of refinement MENTO subbit plane pass, A means to access

the memory which holds the group of a multiplier using the acquired information in order to access only the multiplier identified noting that it was contained in refinement MENTO pass, a means to encode the refinement MENTO bit acquired from memory, and equipment performed as be alike.

[Claim 17] It is equipment according to claim 16 with which two or more refinement MENTO bits are acquired, and a non-refining MENTO bit is disregarded.

[Claim 18] The group of a multiplier is equipment according to claim 16 which constitutes a sign block.

[Claim 19] A means to create the above-mentioned DS is equipment according to claim 16 which has a means to create at least one DS by creating the DS which describes the location of the multiplier of refinement MENTO subbit plane pass to the group of a multiplier.

[Claim 20] It is equipment according to claim 19 with which the 1st DS is created during significant propagation pass for a refinement MENTO bit, and the 2nd DS is created for a clean-up bit.

[Claim 21] DS is equipment containing the indicator of each run of a refinement MENTO bit, and the number of the multiplier which should be skipped within the

group of a multiplier before the run next to a refinement MENTO bit according to claim 19.

[Claim 22] At least for one of the indicator of each run of a refinement MENTO bit, and the numbers of the multiplier which should be skipped, a count 1 is [a symbolic language 000 and a count 2] equipment according to claim 21 with which a symbolic language 001 and a count 3 are expressed by the symbolic language 010, and a count 4 is expressed by a symbolic language 011 and the count 5 with a variable-length sign [like a symbolic language 1111111111011] whose count 4096 is like [a symbolic language 1 trillion and a count 6] a symbolic language billion and the following.

[Claim 23] At least one of the indicator of each run of a refinement MENTO bit and the numbers of the multiplier which should be skipped a count 1 -- a symbolic language 0 and a count 2 -- a symbolic language 01 and a count 3 -- a symbolic language 1100 and a count 4 -- a symbolic language 1101_0 and a count 5 -- a symbolic language 1101_1 and a count 6 -- a symbolic language 1110_0000 and a count 7 -- a symbolic language 1110_0001 -- similarly A count 21 a symbolic language 1110_1111 and a count 22 a symbolic language 1111_0000_0000_0000 and a count 23 like a symbolic language

1111_0000_0000_0001 and the following A count 4096 is equipment according to claim 21 expressed with a variable-length sign like a symbolic language 1111_1111_1110_1010.

[Claim 24] At least one of the indicator of each run of a refinement MENTO bit and the numbers of the multiplier which should be skipped A count 1 the symbolic language 0 of gamma1 format, the symbolic language 0 of gamma format, and a count 2 The symbolic language 10_0 of gamma1 format, the symbolic language 100 of gamma format, and a count 3 The symbolic language 10_1 of gamma1 format, the symbolic language 110 of gamma format, and a count 4 The symbolic language 110_00 of gamma1 format, the symbolic language 10100 of gamma format, and a count 5 The symbolic language 110_01 of gamma1 format, the symbolic language 10110 of gamma format, and a count 6 The symbolic language 110_10 of gamma1 format, the symbolic language 11100 of gamma format, and a count 7 The symbolic language 110_11 of gamma1 format, the symbolic language 11110 of gamma format, and a count 8 The symbolic language 1110_000 of gamma1 format, the symbolic language 1010100 of gamma format, and a count 9 Like the symbolic language 1110_001 of gamma1 format, the symbolic language 1010110 of gamma format, and the

following, a count 15 The symbolic language 1110_111 of gamma1 format, the symbolic language 1111110 of gamma format, and a count 16 The symbolic language 11110_0000 of gamma1 format, the symbolic language 101010100 of gamma format, and a count 32 The symbolic language 111110_00000 of gamma1 format, the symbolic language 10101010100 of gamma format, and a count 64 The symbolic language 111110_000000 of gamma1 format, the symbolic language 1010101010100 of gamma format, and a count 128 The symbolic language 11111110_0000000 of gamma1 format, the symbolic language 101010101010100 of gamma format, and a count 256 The symbolic language 111111110_00000000 of gamma1 format, the symbolic language 10101010101010100 of gamma format, and a count 512 The symbolic language 1111111110_000000000 of gamma1 format, the symbolic language 1010101010101010100 of gamma format, and a count 1024 Symbolic-language 1010101010101010101 00 of the symbolic language 11111111110_000000000 of gamma1 format and gamma format and a count 2048 Symbolic-language 1010101010101010101 0100 of the symbolic language 111111111110_0000000000 of gamma1 format and gamma format and a count 4096 symbolic-language 10101010101010101010 010100 of the symbolic

language 1111111111110_00000000000 of gamma1 format, and gamma format, and ** -- the equipment according to claim 21 expressed with a gamma sign [like].

[Claim 25] The function to perform significant propagation pass about the group of a multiplier, The function which creates the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, DS is accessed in order to acquire the information which identifies the multiplier which should be skipped in the case of processing of refinement MENTO subbit plane pass. In order to access only the multiplier identified noting that it was contained in refinement MENTO pass, The program for making a computer realize the function to perform refinement MENTO subbit plane pass by accessing the memory holding the group of a multiplier using the acquired information, and encoding the refinement MENTO bit acquired from memory.

[Claim 26] The procedure of performing significant propagation pass about the group of a multiplier, The procedure which creates the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, It has the procedure of performing clean-up

subbit plane pass. Clean-up subbit plane pass The procedure which accesses DS in order to acquire the information which identifies the multiplier which should be skipped in the case of processing of clean-up subbit plane pass, The procedure which accesses the memory which holds the group of a multiplier using the acquired information in order to access only the multiplier identified noting that it was contained in clean-up pass, the procedure which encodes the clean-up bit acquired from memory, and the approach performed as be alike.

[Claim 27] It is the approach according to claim 26 by which two or more clean-up bits are acquired, and a non-clean-up bit is disregarded.

[Claim 28] The group of a multiplier is the approach according to claim 26 of constituting a sign block.

[Claim 29] The procedure which creates the above-mentioned DS is the approach according to claim 26 of creating at least one DS by creating the DS which describes the location of the multiplier of clean-up subbit plane pass to the group of a multiplier.

[Claim 30] DS is an approach containing the indicator of each run of a clean-up bit, and the number of the multiplier which should be skipped within the group of a multiplier before the run next to a clean-up bit according to claim 29.

[Claim 31] At least for one of the indicator of each run of a clean-up bit, and the numbers of the multiplier which should be skipped, a count 1 is [a symbolic language 000 and a count 2] an approach according to claim 30 by which a symbolic language 001 and a count 3 are expressed by the symbolic language 010, and a count 4 is expressed by a symbolic language 011 and the count 5 with a variable-length sign [like a symbolic language 1111111111011] whose count 4096 is like [a symbolic language 1 trillion and a count 6] a symbolic language billion and the following.

[Claim 32] At least one of the indicator of each run of a clean-up bit and the numbers of the multiplier which should be skipped a count 1 -- a symbolic language 0 and a count 2 -- a symbolic language 01 and a count 3 -- a symbolic language 1100 and a count 4 -- a symbolic language 1101_0 and a count 5 -- a symbolic language 1101_1 and a count 6 -- a symbolic language 1110_0000 and a count 7 -- a symbolic language 1110_0001 -- similarly A count 21 a symbolic language 1110_1111 and a count 22 a symbolic language 1111_0000_0000_0000 and a count 23 like a symbolic language 1111_0000_0000_0001 and the following A count 4096 is an approach according to claim 30 expressed with a variable-length sign like a symbolic

language 1111_1111_1110_1010.

[Claim 33] At least one of the indicator of each run of a clean-up bit and the numbers of the multiplier which should be skipped A count 1 the symbolic language 0 of gamma1 format, the symbolic language 0 of gamma format, and a count 2 The symbolic language 10_0 of gamma1 format, the symbolic language 100 of gamma format, and a count 3 The symbolic language 10_1 of gamma1 format, the symbolic language 110 of gamma format, and a count 4 The symbolic language 110_00 of gamma1 format, the symbolic language 10100 of gamma format, and a count 5 The symbolic language 110_01 of gamma1 format, the symbolic language 10110 of gamma format, and a count 6 The symbolic language 110_10 of gamma1 format, the symbolic language 11100 of gamma format, and a count 7 The symbolic language 110_11 of gamma1 format, the symbolic language 11110 of gamma format, and a count 8 The symbolic language 1110_000 of gamma1 format, the symbolic language 1010100 of gamma format, and a count 9 Like the symbolic language 1110_001 of gamma1 format, the symbolic language 1010110 of gamma format, and the following, a count 15 The symbolic language 1110_111 of gamma1 format, the symbolic language 1111110 of gamma format, and a count 16 The symbolic language

11110_0000 of gamma1 format, the symbolic language 101010100 of gamma format, and a count 32 The symbolic language 111110_00000 of gamma1 format, the symbolic language 10101010100 of gamma format, and a count 64 The symbolic language 111110_000000 of gamma1 format, the symbolic language 1010101010100 of gamma format, and a count 128 The symbolic language 11111110_0000000 of gamma1 format, the symbolic language 101010101010100 of gamma format, and a count 256 The symbolic language 111111110_00000000 of gamma1 format, the symbolic language 10101010101010100 of gamma format, and a count 512 The symbolic language 1111111110_000000000 of gamma1 format, the symbolic language 1010101010101010100 of gamma format, and a count 1024 Symbolic-language 1010101010101010101 00 of the symbolic language 11111111110_000000000 of gamma1 format and gamma format and a count 2048 Symbolic-language 1010101010101010101 0100 of the symbolic language 111111111110_0000000000 of gamma1 format and gamma format and a count 4096 symbolic-language 1010101010101010101 010100 of the symbolic language 1111111111110_00000000000 of gamma1 format, and gamma format, and ** -- the approach according to claim 30 expressed with a gamma sign

[like].

[Claim 34] The indicator of each run of a clean-up bit is an approach according to claim 30 expressed with a variable-length sign.

[Claim 35] The number of the multiplier which should be skipped is an approach according to claim 30 expressed with a variable-length sign.

[Claim 36] The indicator of each run of a clean-up bit and the number of a multiplier which should be skipped are an approach according to claim 30 expressed with a variable-length sign.

[Claim 37] The indicator of each run of a clean-up bit is an approach according to claim 30 held as an integer.

[Claim 38] An integer is an approach according to claim 37 expressed with the integer of the minimum size.

[Claim 39] The number of the multiplier which should be skipped is an approach according to claim 30 held as an integer.

[Claim 40] An integer is an approach according to claim 39 expressed with the integer of the minimum size.

[Claim 41] The indicator of each run of a clean-up bit and the number of a multiplier which should be skipped are an approach according to claim 30 held

as an integer.

[Claim 42] An integer is an approach according to claim 41 expressed with the integer of the minimum size.

[Claim 43] The procedure which creates DS is the approach according to claim 26 of creating at least one DS by creating the 1st DS which describes the location of the multiplier of the refinement MENTO subbit plane pass to the group of a multiplier, and the 2nd DS which describes the location of the multiplier of clean-up subbit plane pass ** to the group of a multiplier.

[Claim 44] The group of a multiplier is the approach according to claim 43 of constituting a sign block.

[Claim 45] A means to perform significant propagation pass about the group of a multiplier, A means to create the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, It has a means to perform clean-up subbit plane pass. Clean-up subbit plane pass A means to access DS in order to acquire the information which identifies the multiplier which should be skipped in the case of processing of clean-up subbit plane pass, A means to access the memory which holds the group of a multiplier using the acquired information in order to access only the

multiplier identified noting that it was contained in clean-up pass, a means to encode the clean-up bit acquired from memory, and equipment performed as be alike.

[Claim 46] It is equipment according to claim 45 with which two or more clean-up bits are acquired, and a non-clean-up bit is disregarded.

[Claim 47] The group of a multiplier is equipment according to claim 45 which constitutes a sign block.

[Claim 48] A means to create the above-mentioned DS is equipment according to claim 45 which has a means to create at least one DS by creating the DS which describes the location of the multiplier of clean-up subbit plane pass to the group of a multiplier.

[Claim 49] DS is equipment containing the indicator of each run of a clean-up bit, and the number of the multiplier which should be skipped within the group of a multiplier before the run next to a clean-up bit according to claim 48.

[Claim 50] At least for one of the indicator of each run of a clean-up bit, and the numbers of the multiplier which should be skipped, a count 1 is [a symbolic language 000 and a count 2] equipment according to claim 49 with which a symbolic language 001 and a count 3 are expressed by the symbolic language

010, and a count 4 is expressed by a symbolic language 011 and the count 5 with a variable-length sign [like a symbolic language 1111111111011] whose count 4096 is like [a symbolic language 1 trillion and a count 6] a symbolic language billion and the following.

[Claim 51] At least one of the indicator of each run of a clean-up bit and the numbers of the multiplier which should be skipped a count 1 -- a symbolic language 0 and a count 2 -- a symbolic language 01 and a count 3 -- a symbolic language 1100 and a count 4 -- a symbolic language 1101_0 and a count 5 -- a symbolic language 1101_1 and a count 6 -- a symbolic language 1110_0000 and a count 7 -- a symbolic language 1110_0001 -- similarly A count 21 a symbolic language 1110_1111 and a count 22 a symbolic language 1111_0000_0000_0000 and a count 23 like a symbolic language 1111_0000_0000_0001 and the following A count 4096 is equipment according to claim 49 expressed with a variable-length sign like a symbolic language 1111_1111_1110_1010.

[Claim 52] At least one of the indicator of each run of a clean-up bit and the numbers of the multiplier which should be skipped A count 1 the symbolic language 0 of gamma1 format, the symbolic language 0 of gamma format, and a

count 2 The symbolic language 10_0 of gamma1 format, the symbolic language 100 of gamma format, and a count 3 The symbolic language 10_1 of gamma1 format, the symbolic language 110 of gamma format, and a count 4 The symbolic language 110_00 of gamma1 format, the symbolic language 10100 of gamma format, and a count 5 The symbolic language 110_01 of gamma1 format, the symbolic language 10110 of gamma format, and a count 6 The symbolic language 110_10 of gamma1 format, the symbolic language 11100 of gamma format, and a count 7 The symbolic language 110_11 of gamma1 format, the symbolic language 11110 of gamma format, and a count 8 The symbolic language 1110_000 of gamma1 format, the symbolic language 1010100 of gamma format, and a count 9 Like the symbolic language 1110_001 of gamma1 format, the symbolic language 1010110 of gamma format, and the following, a count 15 The symbolic language 1110_111 of gamma1 format, the symbolic language 1111110 of gamma format, and a count 16 The symbolic language 11110_0000 of gamma1 format, the symbolic language 101010100 of gamma format, and a count 32 The symbolic language 111110_00000 of gamma1 format, the symbolic language 10101010100 of gamma format, and a count 64 The symbolic language 111110_000000 of gamma1 format, the symbolic

language 1010101010100 of gamma format, and a count 128 The symbolic language 11111110_0000000 of gamma1 format, the symbolic language 101010101010100 of gamma format, and a count 256 The symbolic language 111111110_00000000 of gamma1 format, the symbolic language 10101010101010100 of gamma format, and a count 512 The symbolic language 1111111110_000000000 of gamma1 format, the symbolic language 1010101010101010100 of gamma format, and a count 1024 Symbolic-language 1010101010101010101 00 of the symbolic language 11111111110_000000000 of gamma1 format and gamma format and a count 2048 Symbolic-language 1010101010101010101 0100 of the symbolic language 111111111110_0000000000 of gamma1 format and gamma format and a count 4096 symbolic-language 10101010101010101010 010100 of the symbolic language 1111111111110_00000000000 of gamma1 format, and gamma format, and ** -- the equipment according to claim 49 expressed with a gamma sign [like].

[Claim 53] The indicator of each run of a clean-up bit is an approach according to claim 49 expressed with a variable-length sign.

[Claim 54] The number of the multiplier which should be skipped is an approach

according to claim 49 expressed with a variable-length sign.

[Claim 55] The indicator of each run of a clean-up bit and the number of a multiplier which should be skipped are an approach according to claim 49 expressed with a variable-length sign.

[Claim 56] The function to perform significant propagation pass about the group of a multiplier, The function which creates the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, In order to access only the multiplier identified noting that DS was accessed and it was contained in clean-up pass, in order to acquire the information which identifies the multiplier which should be skipped in the case of processing of clean-up subbit plane pass, The program for making a computer realize the function to perform clean-up subbit plane pass by accessing the memory holding the group of a multiplier using the acquired information, and encoding the clean-up bit acquired from memory.

[Claim 57] The refinement MENTO bit run count which specifies the size of a run of a refinement MENTO bit, The refinement MENTO bit skip count which specifies the number of the multiplier which should be skipped between the runs of a refinement MENTO bit, The clean-up bit run count which specifies the size

of a run of a clean-up bit, And the clean-up bit skip count which specifies the number of the multiplier which should be skipped between the runs of a clean-up bit is held. The procedure of directing the location of the multiplier within the group of the multiplier which should be processed between refinement MENTO pass or clean-up pass and that a context model reads at least one DS, How to have the procedure in which a context model accesses memory based on the information within at least one DS.

[Claim 58] Memory and the context model connected to memory are provided. A context model The refinement MENTO bit run count which specifies the size of a run of a refinement MENTO bit, The refinement MENTO bit skip count which specifies the number of the multiplier which should be skipped between the runs of a refinement MENTO bit, The clean-up bit run count which specifies the size of a run of a clean-up bit, And the clean-up bit skip count which specifies the number of the multiplier which should be skipped between the runs of a clean-up bit is held. Direct the location of the multiplier within the group of the multiplier which should be processed between refinement MENTO pass or clean-up pass. It is the decoder with which at least one DS is read and a context model accesses memory based on the information within at least one DS.

[Claim 59] In order to acquire the information which identifies the multiplier which should be skipped in the case of processing of refinement MENTO subbit plane pass, In order to access only the procedure which accesses the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, and the multiplier identified noting that it was contained in refinement MENTO pass, How to have the procedure which accesses the memory holding the group of a multiplier using the acquired information, and the procedure which encodes the refinement MENTO bit acquired from memory.

[Claim 60] The method according to claim 59 of having further the procedure of performing significant propagation pass by encoding the multiplier of significant propagation pass while creating the above-mentioned DS.

[Claim 61] In order to acquire the information which identifies the multiplier which should be skipped in the case of processing of refinement MENTO subbit plane pass, In order to access only a means to access the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, and the multiplier identified noting that it was contained in refinement MENTO pass, Equipment which has a means to access

the memory holding the group of a multiplier using the acquired information, and a means to encode the refinement MENTO bit acquired from memory.

[Claim 62] Equipment according to claim 59 which has further a means to perform significant propagation pass including a means to encode the multiplier of significant propagation pass while creating the above-mentioned DS.

[Claim 63] In order to acquire the information which identifies the multiplier which should be skipped in the case of processing of clean-up subbit plane pass, In order to access only the procedure which accesses the DS which shows the location of the multiplier within the group of the multiplier which should be processed with consecutive pass, and the multiplier identified noting that it was contained in clean-up pass, How to have the procedure which accesses the memory holding the group of a multiplier using the acquired information, and the procedure which encodes the clean-up bit acquired from memory.

[Claim 64] The method according to claim 63 of having further the procedure of performing significant propagation pass by encoding the multiplier of significant propagation pass while creating the above-mentioned DS.

[Claim 65] Equipment which decrypts the information possessing the memory which has the 1st part holding a run count, the 2nd part holding a skip count, and

the 3rd part that separates the 1st part and 2nd part, and the decoder which decrypts the run count and the skip count which were connected to the above-mentioned memory and were simultaneously gained from the above-mentioned memory.

[Claim 66] The 3rd part of the above-mentioned memory is equipment containing the part for which the memory between the 1st part of the above and the 2nd part of the above is not used according to claim 65.

[Claim 67] The part for which the above-mentioned memory is not used is equipment according to claim 66 which adjoins the 1st part of the above, and the 2nd part of the above.

[Claim 68] Are the approach of performing significant propagation pass and the information processed according to the significant condition information over a field Significant propagation pass, refinement MENTO pass, The procedure to judge and the sub field in a field whether it belongs to clean-up pass Or significant propagation pass, refinement MENTO pass, Or while discriminating the run count and skip count to refinement MENTO pass and clean-up pass from the procedure which asserts the signal showing whether it has the multiplier of clean-up pass How to have the procedure of processing the bit of the multiplier

authorized as a thing on the bit and clean-up pass of a multiplier which were authorized as the procedure which encodes the bit of the multiplier of significant propagation pass, and a thing on refinement MENTO pass.

[Claim 69] The method according to claim 68 of a precedence bit being a significant propagation bit, and having further the procedure which starts the new run to either refinement MENTO pass or clean-up pass, when a current bit is not a significant propagation bit.

[Claim 70] The method according to claim 69 of having further the procedure of adjusting the index to the table within the DS holding run count information and skip count information to a new entry, and the procedure which initializes a new entry to the condition of expressing initiation of a new run.

[Claim 71] The procedure which initializes a new entry is the approach according to claim 70 of initializing a skip flag to zero and initializing a run indicator to a certain value.

[Claim 72] Each sub field is an approach according to claim 68 constituted by 4x4 fields.

[Claim 73] The method according to claim 68 of having further the procedure which counts the number of a significant multiplier in each sub field.

[Claim 74] The method according to claim 68 of having further the procedure of processing all the bits in each sub field as refinement MENTO, when a count is in agreement with the size of each sub field.

[Claim 75] The approach according to claim 68 a count has further the procedure of processing all the bits in each sub field as clean-up when significant near does not exist in accordance with zero.

[Claim 76] Are equipment which performs significant propagation pass and the information processed according to the significant condition information over a field Significant propagation pass, refinement MENTO pass, A means to judge, and the sub field in a field whether it belongs to clean-up pass Or significant propagation pass, refinement MENTO pass, Or while discriminating the run count and skip count to refinement MENTO pass and clean-up pass from a means to assert the signal showing whether it has the multiplier of clean-up pass Equipment which has a means to process the bit of the multiplier authorized as a thing on the bit and clean-up pass of a multiplier which were authorized as a means to encode the bit of the multiplier of significant propagation pass, and a thing on refinement MENTO pass.

[Claim 77] Equipment according to claim 76 which a precedence bit is a

significant propagation bit, and has further a means to start the new run to either refinement MENTO pass or clean-up pass when a current bit is not a significant propagation bit.

[Claim 78] Equipment according to claim 77 which has further a means to adjust the index to the table within the DS holding run count information and skip count information to a new entry, and a means to initialize a new entry to the condition of expressing initiation of a new run.

[Claim 79] A means to initialize a new entry is equipment according to claim 78 which initializes a skip flag to zero and initializes a run indicator to a certain value.

[Claim 80] Each sub field is equipment according to claim 76 constituted by 4x4 fields.

[Claim 81] Equipment according to claim 76 which has further a means to count the number of a significant multiplier in each sub field.

[Claim 82] Equipment according to claim 76 which has further a means to process all the bits in each sub field as refinement MENTO when a count is in agreement with the size of each sub field.

[Claim 83] Equipment according to claim 76 with which a count has further a means to process all the bits in each sub field as clean-up when significant near

does not exist in accordance with zero.

[Claim 84] The input which is equipment which performs significant propagation pass and receives the significant condition information over the 1st field of the 1st predetermined size, In a list [whether the pass relevant to each multiplier in the 2nd field of the 2nd predetermined size is significant propagation pass or it is refinement MENTO pass, and] Or the decision pass unit possessing the group of the 1st output indicator in which it is shown whether it is clean-up pass, The processing unit possessing the input connected to the output indicator, the next non-refining MENTO bit output, the next non-clean-up bit output, and a current pass indicator output, Connect with an output from a processing unit and it responds to an output from the above-mentioned processing unit. The control unit which generates the indicator, the refinement MENTO run indicator, the next refinement MENTO skip flag, and next significant propagation indicator of an index to refinement MENTO bit-data structure, and equipment with which ** was prepared.

[Claim 85] The above-mentioned processing unit the count showing the group of the 1st output indicator, and the current multiplier of the 2nd field currently processed Reception, The selection unit with which the group of the 2nd [to a

current multiplier] output indicator is provided, and only one output indicator in the group of the 2nd output indicator is asserted to the current multiplier, The refinement MENTO pass indicator and clean-up pass indicator, and count from a decision pass unit Reception, By carrying out the mask of the bit of the refinement MENTO pass indicator relevant to the multiplier in the 2nd already processed field, and a clean-up pass indicator The mask unit which has the pair of the output expressing a mask mold refinement MENTO pass indicator and a mask mold clean-up pass indicator, The input which receives a mask mold refinement MENTO pass indicator and a mask mold clean-up pass indicator is provided. Equipment according to claim 84 which has a priority encoder possessing the pair of the output expressing the next non-refining MENTO bit position to significant propagation pass, and the next non-clean-up bit position.

[Claim 86] A control unit is equipment according to claim 84 which answers an output from a processing unit and generates the indicator, the next clean-up run indicator, and next clean-up skip flag of an index to clean-up bit-data structure.

[Claim 87] It is equipment according to claim 84 by which a multiplier is processed with significant propagation pass when a current bit belongs to significant propagation pass.

[Claim 88] Equipment according to claim 84 by which K bits which are in agreement with K which is the run length of refinement MENTO pass when a current bit belongs to refinement MENTO pass are processed with refinement MENTO pass.

[Claim 89] Equipment according to claim 86 by which K bits which are in agreement with K which is the run length of clean-up pass when a current bit belongs to clean-up pass are processed with clean-up pass.

[Claim 90] It is equipment according to claim 84 which a control unit sets up the refinement MENTO run indicator which is in agreement with a run length, and sets up a refinement MENTO skip flag so that the purport to which a skip of the multiplier in the 2nd field should not be carried out based on a current multiplier may be directed when the following non-refining MENTO bit should be processed later in time than a current multiplier.

[Claim 91] A control unit is equipment according to claim 90 which sets up the next index indicator of refinement MENTO so that the purport to which the index to the DS holding the run count and skip count which direct the location of the refinement MENTO bit within a sign block should be made to increase may be directed.

[Claim 92] When the following non-refining MENTO bit should be processed later in time than a current multiplier, a control unit The refinement MENTO run indicator which is in agreement with a run length is set up. A skip of the multiplier in the 2nd field A refinement MENTO skip flag is set up so that the purport which should not be performed based on a current multiplier may be directed. Further a control unit The next index indicator of refinement MENTO is set up so that the purport to which the index to the DS holding the run count and skip count which direct the location of the refinement MENTO bit within a sign block should be made to increase may be directed. Further a control unit The clean-up run indicator which directs the purport in which the current run of a clean-up bit has not appeared is set up. Equipment according to claim 86 which sets up the clean-up skip flag which is in agreement with the value which deducted the number equal to the location of the current multiplier in the 2nd field from the next non-refining MENTO bit position.

[Claim 93] A control unit is equipment according to claim 92 which sets up the next index indicator of clean-up so that the purport which should not change the index to the DS holding the run count and skip count which direct the location of the clean-up bit within a sign block may be directed.

[Claim 94] A significant propagation index is equipment according to claim 93 set up so that the purport by which a current multiplier does not belong to significant propagation pass may be shown.

[Claim 95] It is equipment according to claim 86 which a control unit sets up the clean-up run indicator which is in agreement with a run length, and sets up a clean-up skip flag so that the purport to which a skip of the multiplier in the 2nd field should not be carried out based on a current multiplier may be directed when the following non-clean-up bit should be processed later in time than a current multiplier.

[Claim 96] A control unit is equipment according to claim 95 which sets up the next index indicator of clean-up so that the purport to which the index to the DS holding the run count and skip count which direct the location of the clean-up bit within a sign block should be made to increase may be directed.

[Claim 97] A control unit is equipment according to claim 96 which sets up the clean-up run indicator which directs further the purport in which the current run of a clean-up bit has not appeared, and sets up the clean-up skip flag which is in agreement with the value which deducted the number equal to the location of the current multiplier in the 2nd field from the next non-refining MENTO bit position.

[Claim 98] A control unit is equipment according to claim 97 which sets up the next index indicator of clean-up so that the purport to which the index to the DS holding the run count and skip count which direct the location of the clean-up bit within a sign block should not be changed may be directed.

[Claim 99] A significant propagation index is equipment according to claim 98 set up so that the purport by which a current multiplier does not belong to significant propagation pass may be shown.

[Claim 100] A decision pass unit is equipment according to claim 84 which inspects a predetermined number per piece of Nx M regions for every multiplier processed when N and M express an integer.

[Claim 101] Nx M region is equipment according to claim 100 constituted by 3x3 fields.

[Claim 102] The predetermined number of Nx M regions is equipment according to claim 100 whose number is 16.

[Claim 103] For each output indicator of significant propagation pass, refinement MENTO pass, and clean-up pass, the group of the 1st output indicator is equipment according to claim 86 which is 16-bit width of face including the output indicator of significant propagation pass, refinement MENTO pass, and

clean-up pass.

[Claim 104] Each output indicator of significant propagation pass, refinement MENTO pass, and clean-up pass is equipment according to claim 86 with which it has 1 bit corresponding to the multiplier of the 2nd field, and only the only bit in it is asserted in each cycle.

[Claim 105] A mask unit is equipment according to claim 86 which carries out the mask of the signal line of each indicator of the refinement MENTO pass corresponding to the already processed multiplier, and clean-up pass.

[Claim 106] A priority encoder is equipment containing the priority encoder for null detection according to claim 85.

DETAILED DESCRIPTION

[Detailed Description of the Invention]

[0001]

[Field of the Invention] This invention relates to the technique of compression and extension, and relates to the context model skipping technique and memory access technique based on a run skip count especially.

[0002]

[Description of the Prior Art] A data compression is a very effective tool for storing and transmitting the data of a large quantity. For example, the time amount required in order to transmit an image like the facsimile transmission of a document will be dramatically reduced, if compression is used in order to decrease the number of bits required in order to show an image.

[0003] Various data compression techniques are known for the conventional technique. Compression technology is divided roughly into two categories, coding with a loss and coding without a loss. Coding which produces loss of information perfect reconstruction of original data is not guaranteed to be is included in coding with a loss. The target of lossy compression is carried out so that modification to original data may not become obstructive or it cannot detect. It is compressing data in lossless compression, to hold all information and to be able to perform perfect reconstruction.

[0004] In lossless compression, an input symbol or data on the strength is changed into an output symbolic language. An input contains an image, an audio, 1-dimensional data (for example, data which change spatially), two-dimensional data (for example, data which change by the spatial 2-way), or many dimensions / multiplex spectrum data. When compression is successful, a symbolic language is expressed in bits fewer than the number of bits in the "Normal" expression of an input signal (or data on the strength). The coding approach without a loss includes the encoding method (for example, Lempel-Ziv law) like a dictionary, a run-length-coding method, the ENYUMERATIBU (enumerative) encoding method, and an entropy-code-modulation method. In the case of

picture compression without a loss, compression is due to prediction, or a context and coding. The standard [for JBIG] one for facsimile compression and DPCM for continuous tone images (one option in a differential-pulse-code-modulation-JPEG criterion) are the examples of the lossless compression for images. In lossy compression, an input symbol or data on the strength is quantized before changing into an output symbolic language. It has the intention of quantization so that the property of relevant data may be saved, while removing the property which is not important. Conversion is often used for the compression system which has loss before quantization in order to perform energy concentration. JPEG is an example of the coding approach with loss for image data.

[0005] Invertible transformation (wavelet transform, component conversion) is used for compression of both lossy compression and lossless compression. Irreversible conversion (wavelet, a component, discrete cosine) is used only for lossy compression.

[0006] A new JPEG 2000 decryption criterion uses conversion, and offers the new coding system for images, and a sign stream convention. Although JPEG 2000 criterion is a decryption criterion and the form which should have a decoder

is defined, especially this definition restrains the encoder without a loss for compression. Each image is divided into a rectangle tile under JPEG 2000 criterion. A tile component is generated by the tiling of an image when two or more tiles exist. An image contains many components. For example, a color picture has a component, which are red, green, and blue. A tile component is extracted or decrypted independently of mutual.

[0007] A tile component is disassembled into one or more different decomposition level after the tiling of an image using wavelet transform. Such decomposition level holds the subband of a large number which it had by the multiplier which describes the horizontal and spatial frequency characteristics of a original tile component. A multiplier gives not the whole image but the frequency information about a partial field. That is, a sample with a small number of single multiplier is described thoroughly. Decomposition level is related with the following decomposition level for the spatial scale factor 2 so that the horizontal resolution and vertical resolution of decomposition level which the subband followed may become abbreviation half [of front decomposition level].

[0008] Although the multiplier of a sample and the same number exists, it is tended to concentrate the content of information only on some multipliers. By

quantization, the numerical precision of many multipliers decreases with installation of a low distortion (quantization noise) to unbalance. The additional processing by the entropy encoder decreases the number of bits demanded in order to express these quantized multipliers, and is decreased more remarkably than a subject-copy image depending on the case.

[0009] The subband according to individual of tile MPONENTO is further divided into a sign block. Grouping of the sign block is carried out to an area. The rectangle-like array of these multipliers may be extracted independently. Entropy code modulation of the bit plane according to individual of the multiplier under sign block is carried out with the coding pass of a three-stage. Each coding pass of the three-stages collects the context (context) information about bit plane compression image data.

[0010] The group division of the bit stream compression image data generated from these coding pass is carried out at a layer. A layer is a group division of the arbitration of the coding pass which continued from the sign block. Although layer-ization is rich in flexibility, it is a premise that each continuous layer contributes to the image of high quality more. each -- resolving power -- the sign block of the subband multiplier in level is divided into the rectangle field called an

area (precincts).

[0011] A packet is the base unit of a compression sign stream. a packet -- one resolving power of one tile component -- the compression image data from one prediction layer of level is included. These packets are arranged in the sequence defined into the sign stream.

[0012] The sign stream relevant to a tile is constituted by the packet, and is put in order by one or more tile parts. A tile partial header is constituted by a marker and a marker segment, or the sequence of a tag, and holds the information about various structure and coding styles required in order to find, extract, decode and reconstruct the location of all tile components. The main header constituted by the marker and the marker segment is at the head of all sign streams, and the main header offers information similar to the information list about a subject-copy image.

[0013] A sign stream is summarized to a file format in optional so that application can interpret the semantics of an image, and the semantics of the information on others about an image. This file format holds data other than a sign stream.

[0014] A decryption of a JPEG 2000 sign stream is performed by reversing the sequence of a coding procedure. The block diagram of a JPEG 2000 standard

coding system which acts on a compression image data sign stream is shown in drawing 1 . With reference to drawing 1 , first, a bit stream is received by the data sequencing block 101, and the data sequencing block 101 carries out the re-group division of a layer and the subband multiplier. The multiplier about the bit plane compression image data which was encoded before and which was given from the bit modeling block 103 and the context information from the internal state of the bit modeling block 103 are used for the algebraic-sign machine 102, and it decrypts a compression bit stream.

[0015] Next, a sign stream is quantized by the quantization block 104, and the quantization block 104 quantizes based on the improvement processing (ROI) in field selection image quality as shown by the ROI (improvement in field selection image quality) block 105. Reverse wavelet / space conversion is applied to a multiplier by the conversion block 106 after quantization, and DC level shift and the alternative component conversion block 107 continue after that. Thereby, a playback image is generated.

[0016]

[Problem(s) to be Solved by the Invention] This invention aims information the approach list encoded and decrypted at offer of equipment.

[0017]

[Means for Solving the Problem] In one example, the equipment of this invention contains memory and decryption hardware. Memory accumulates a run count and/or a skip count, and decryption hardware decrypts a run count and/or the skip count acquired from memory during the decryption.

[0018] This invention will fully be understood by referring to a publication and accompanying drawing of the following examples. However, a publication and accompanying drawing of an example do not limit this invention to a specific example, and mean explanation and a break through of this invention.

[0019]

[Embodiment of the Invention] The technique of performing coding is explained. The technique of performing coding is used in order to realize JPEG 2000 criterion and to operate the description set, or in order to add to the description set. Namely, ITU-T Rec.T.800|ISO/IEC FDIS 15444 quoted as JPEG 2000 criterion for reference: Core Coding System (core coding system) which is the information technology-JPEG 2000 image coding system indicated by 2000 JPEG Image Coding System leaves an introductory person the range of much selections. In software, hardware, and/or firmware, the object of the technique

indicated by the above-mentioned reference is the thing using the range of selection of JPEG 2000 in order to realize a high speed, a low price, small memory space, and/or a means with abundant engine performance.

[0020] Below, much details are explained so that this invention can understand thoroughly. However, probably, it will be clear to this contractor to carry out without this invention using these specific details. It is shown by in the form of the block diagram instead of detail drawing in order, as for a well-known configuration and equipment, to avoid that the essence of this invention becomes unclear in the case of others. Furthermore, it realizes using a well-known means, or the block, the logic, or the function which is not indicated by the detail is easily realized by this contractor using hardware, well-known software, and/or well-known firmware. It is necessary to notice some of techniques and means about being described using a pseudocode. However, this does not necessarily mean that some of these techniques and means are realized only by software, and rather, such a publication is often chosen, in order to explain briefly the function of the item which this contractor will understand easily.

[0021] A part of following detailed explanation is shown using the algorithm of an

operation and notation-expression to the data bit in a computer memory. These algorithm-description and expressions are means to use it in order that this contractor of the field of a data-processing technique may tell this contractor of the field his achievements most efficiently. Here, it is thought that an algorithm is generally the sequence of the consistent procedure which draws the result of a request. These procedures are procedures which need practical actuation of physical quantity. Usually, although it is not necessarily indispensable, such physical quantity is obtained in the form of storage, a transfer, composition, a comparison, the electrical signal that can be operated, or a magnetic signal. Mainly since it is general direction for use, these signals are understood that it is convenient to specify with a bit, a value, an element, a symbol, an alphabetic character, a term, a number, etc. by the way.

[0022] However, these vocabulary and synonyms are only the expedient labels that were related with suitable physical quantity and given to physical quantity. clear from the following explanation, unless it refuses especially -- as -- "processing", "computing", "count", and "decision" -- or The description using vocabulary like "a display" is what shows actuation and processing of a computer system or a similar electronic computing device. The data expressed

as an amount of physics (electron) within the register of a computer system and memory are operated. Similarly The memory or the register of a computer system, Or it changes into other data expressed as physical quantity within other information storage devices, information-transmission equipment, or an information display.

[0023] This invention relates to the equipment which performs actuation explained below. Especially this equipment contains the general purpose computer reconfigurated using the computer program which was constituted according to the demanded application, operated selectively, or was memorized by the computer. Although such a computer program is memorized by the storage which was connected to a computer system bus like a disk, a read-only memory (ROM), random access memory (RAM), EPROM, EEPROM, the MAG or optical card of a type of the arbitration containing a flexible disk, an optical disk, CD-ROM, a magneto-optic disk, etc., and the medium of the type of arbitration which was suitable in order to memorize an electronic instruction and in which computer reading is possible, it is not restricted to these instantiation-storages.

[0024] The algorithm and display which are explained here were not essentially

related with a specific computer or other equipments. Various general-purpose systems are used with the program according to the matter taught here. Or it may be more convenient to constitute the equipment dedication-ized more, in order to perform the procedure of the demanded approach. The configuration demanded from these various systems becomes clear from the following publications. Furthermore, this invention is not explained based on specific programming language. It will be admitted that various programming language can be used in order to realize the instruction matter of this invention which is explained below.

[0025] The medium in which machine reading is possible includes the mechanism of the arbitration which memorizes or transmits information with a machine (for example, computer) in the format which can be read. For example, the medium in which machine reading is possible includes a read-only memory (ROM), random access memory (RAM), a magnetic-disk storage, an optical storage, flash memory equipment, and electric, optical, acoustical or the propagation signals of other formats (for example, a subcarrier, an infrared signal, a digital signal, etc.).

[0026] [Outline] Drawing 29 is the block diagram of one example of an encoder.

With reference to drawing 29 , the data interface (I/F) 2901 is connected in order to receive the data which should be encoded, or the data which should be outputted after a decryption. It connects with the data interface 2901 and the DC level shifter 2902 performs DC level shift during coding and a decryption. It connects with the DC level shifter 2902, and the wavelet transform machine 2903 performs order wavelet transform or reverse wavelet transform depending on the direction of a processing flow. In one example, the wavelet transform machine 2903 performs 5, 3-reversible wavelet transform and 5, and 3-irreversible wavelet transform, and disassembles an image into 2 thru/or 5 level. In case it connects with the wavelet transform machine 2903 and a line buffer 2904 performs wavelet transform, it supplies data to the wavelet transform machine 2903.

[0027] It connects with the wavelet transform machine 2903, and the formation of a scalar quantity child / reverse quantization block 2905 performs scalar quantity child-ization. In one example, scalar quantity child-ization is used only for 5 and 3-wavelet transform. It connects with the scalar quantity child-ized block 2905, and PURIKODA 2906 performs PURIKODINGU. In one example, PURIKODA 2906 changes a multiplier into a scale with a sign (magnitude) from

a two's complement (in a decryption, it transforms inversely). PURIKODA determines a zero bit plane. The work-piece memory A and the work-piece memory B are connected to PURIKODA 2906 with the packet header treater 2907. The interface to the work-piece memory A and the work-piece memory B, and the packet header treater 2907 is connected also to bit modeling MQ encoder 29081-N. Each MQ encoder 29081-N is connected to sign memory 2911N of the individual exception holding coded data (JPEG the vocabulary of 2000 compressed data). The coded data from sign memory and the packet header from the packet header treater 2907 are outputted as coded data. This is JPEG 2000 bit stream. Additional functional block (not shown) is used for creation/read-out of the main header and a tile section header. A bit stream and a header form a JPEG 2000 sign stream.

[0028] [context model DS including the skip to a subbit plane] -- in JPEG 2000, each multiplier is encoded to the bit plane of the multiplier which are not all zero in first stage to a lower bit from the most significant bit (MSB) -- beginning -- by one of significant propagation and refinement MENTO and three subbit plane pass of clean-up. An example of the 8x8 sign block with which the subbit plane pass of one bit plane was identified is shown in drawing 2 A for every multiplier.

When drawing 2 A is referred to, SP expresses significant propagation pass, R expresses refinement MENTO pass, and C expresses clean-up pass. The characteristics 0-63 in drawing 2 A show sign block scan sequence. Thus, scan sequence continues and repeats actuation of returning downward at four multipliers, next the top line, to the whole sign block. After the scan of the whole sign block is completed next, from the 5th multiplier of each train, a scan continues downward throughout the remaining sign block, and is continued.

[0029] In the typical example of implementation, the whole block of a multiplier is read every 1 time [a total of 3] for every coding pass of a bit plane. This technique explains how to read the whole multiplier block to the significant propagation pass of each bit plane, and read only the multiplier actually needed to refinement MENTO pass and clean-up pass.

[0030] In the left-hand side of each cel of drawing 2 A, a continuous line shows the multiplier of refinement MENTO subbit plane pass, and a dotted line expresses the multiplier skipped with refinement MENTO subbit plane pass. The continuous line on the right-hand side of each cel of drawing 2 A shows the multiplier to clean-up subbit plane pass. A multiplier will be processed if pass is identified for every multiplier.

[0031] DS is built during significant propagation pass using the below-mentioned processing. Since this DS reduces the count of access to memory, it is used with a context model by it. What is necessary is to inspect for every cel in order to judge which pass the pass with which information belongs is, and to access a context model only once to having skipped the cel by using this DS, at memory. Furthermore, this DS makes it possible to access simultaneously to many locations like [in case only 4 bits for example, of clean-up bits are encoded simultaneously].

[0032] A table 1 and a table 2 describe the location of the location of the multiplier in refinement MENTO subbit plane pass, and the multiplier in clean-up subbit plane pass, respectively. The run count of the number of the multiplier in subbit plane pass and the skip count of the number of the consecutiveness multiplier contained in different pass are shown for every index. Such DS makes it possible to encode subbit plane pass efficiently. Therefore, it enables it to skip the multiplier in other pass.

[0033]

[A table 1]

表1
リファインメントビット用データ構造

インデックス	ラン	スキップ
0	0	1
1	2	8
2	1	3
3	1	48

[0034]

[A table 2]

This processing is performed by hardware, software, or processing logic including both combination. In one example, DS is created and this processing logic that uses DS is prepared in bit MODERUNGU MQ encoder 29081-N of drawing 29 . In order to create such DS during processing, DS is initialized first.

[0035]

ri=0 // index for refinement MENTO ci=0 // index for clean-up r_run[ri] =0 // run count for refinement MENTO r_skip[ri] =0 // skip count for refinement MENTO

c_run[ci] =0 // run count for clean-up c_skip[ci] =0 // skip count for clean-up state=INITIAL // condition state is INITIAL, SIG_PROP, REFINE, or CLEANUP.

[0036] A state variable is used in order to distinguish initiation and the center of activation (run). A state variable shows the coding pass for front multipliers. When a current multiplier is the same, the size of a run or a skip increases, and a new run is started when it differs. Each multiplier of a sign block is taken into consideration in order of a sign block scan, in order to generate a separate count.

[0037]

For y1=0 to maximum-for-y1 step 4 for x=0 to maximum-for-x step 1 for y2=0 to maximum-for-y2 step 1 In the process coefficient[x, y1+y2] above-mentioned code, the maximum of y1 is the greatest integral multiple of 4 smaller than the height ("(height-1) &-3") of a sign block. The maximum of x is "width-1" of a sign block. The maximum of y2 is the smaller one of 3 and "height-y 1-1." One example of the procedure which processes each multiplier is as follows.

[0038]

A multiplier sets to a front bit plane. if Significant then if then whose state is not REFINE ri=ri +1 r_run[ri] =1 r_skip[ri] =0 state=REFINE else r_run[ri] =r_run [ri] +1 c_skip[ci] =c_skip [ci] +1 else if then significant near the multiplier (significant

sign the multiplier was predicted to be)

r_skip[ri] =r_skip [ri] +1 c_skip[ci] =c_skip [ci] +1 state=SIG_PROP else then whose state is not CLENUP ci=i +1 c_run{ci]=1 c_skip[ci] =0 state=CLEANUP else c_run [ci] =c_run[ci]+1 r_skip[ri] =r_skip[ri] +1-//.

[0039] As a result of applying this procedure, the total multiplier of significant propagation pass is encoded and the DS over a refinement MENTO bit and a clean-up bit is created.

[0040] A run count can be prevented from returning exceeding a line if needed. One example of the processing for preventing the wrap around of this line is indicated by the following pseudocodes. Thereby, it becomes possible to deal with a boundary very simply.

[0041]

[0042] In the case of software, it is most convenient to hold a run value and a skip value as integers (the case of a 32-bit computer 32 etc. bits etc.). The worst case is the run of the die length 1 started by the run of die length 0. In JPEG 2000, the multiplier of a sign block is restricted to 4096 pieces at the maximum. The width of face and the height of a sign block are also restricted to the multiplier of 1024 pieces at the maximum. when it all comes out and a run count continues throughout the group of a line to 4096 sign blocks of the size of the arbitration of a multiplier, 4097 memory locations are the maximum numbers of the memory location to the memory size. When a run count starts for every group of every four lines to the sign block of 64x64, a x(4x64+1) (64/4) =4112 piece memory location is maximum. When a run count starts for every group of every four lines to the sign block of 1024x4, a x(4x4+1) (1024/4) =4352 piece memory location is maximum.

[0043] Since the memory of hardware is saved, and the bits of the minimum number of immobilization are a run count and a skip count, it may be used. When the indicator in which it is shown which [of a run count and a skip count] is the first count is shown by signal (for example, 1-bit signal indicator), a run

count is larger than 1 (and the capacity which encodes 0 is not required). When a run count continues to the group of a line (line) to the sign block of the size of arbitration which has the multiplier of 4096 pieces in all, in order to show whether the first count is a run or it is a skip, 1 bit is used and 4096x12 bits is used to 49,153 bit of the whole. When a run count starts for every group of every four lines to a 64x64 sign block, 1 bit is used in order to show whether the first count is a run or it is a skip to each group of every four lines. In this way, the number of bits is 1x64/4+4096x12=49,168 bit. When a run count starts for every group of four lines to the sign block of 1024x3, the number of bits is 1x1024/4+4096x12=49,408 bit.

[0044] Since a count is expressed, one example of a variable-length sign may be used. In a table 3, a small count is expressed in a small number of bit (for example, triplet), and a large count is expressed by many bits (for example, 13 bits). The target of such approach is expressing almost all counts with 1, 2, 3, or 4 so that a comparatively small symbolic language can be used frequently. Only two sizes are used in order to realize more simply. However, three or more sizes may be used and complexity may be increased.

[0045]

[A table 3]

In the case of this sign, when all run lengths are set to 1 (that is, all symbolic languages triplet), the worst situation arises. The total number of bits becomes 12,289 bits, 12,304 bits, and 12,544 bits to three kinds of situations (the situation which crosses and counts a line, the situation of the 64x64 sign block by the group of every four lines, situation of the 1024x4 sign block by the group of every four lines), respectively.

[0046] The cutback of memory usage is attained by using a more complicated variable-length sign. The outstanding sign of structure is a gamma sign, gamma 1, or gamma (BBell, Cleary, Whitten "Text Compression", Prenice Hall, NJ, 1990 Appendix A) as shown in a table 4.

[0047]

[A table 4]

As for gamma1 and gamma, only arrangement of the bit of a symbolic language is different. "_" in gamma1 symbolic language is not a part of symbolic language, and is for making it legible by separating a prefix from a counter. In the count of 2 which needs a triplet, the worst situation is produced. The total number of bits needed becomes 6,145 bits, 6,160 bits, and 6,400 bits to three kinds of situations (the situation which crosses and counts a line, the situation of the 64x64 sign block by the group of every four lines, situation of the 1024x4 sign block by the group of every four lines), respectively.

[0048] A table 5 is one example of the sign to the count 1...4096 to which the longest symbolic language is expressed with 16 bits. Counts 1, 2, 3, and 4 or 5 is

expressed by 1, 2, 3, 4, and 5, respectively. Counts 6-21 are expressed with 8 bits. Counts 22-4096 are expressed with 16 bits. The count of 3 and 6 is in the worst situation. The total numbers of bits are 5,463 bits, 5,478 bits, and 5,718 bits to three kinds of situations (the situation which crosses and counts a line, the situation of the 64x64 sign block by the group of every four lines, situation of the 1024x4 sign block by the group of every four lines), respectively.

[0049]

[A table 5]

When using the variable-length sign by hardware, it is desirable to access to both a run count and a skip count into the same clock cycle. In one example, since memory is larger than the minimum size by one word (for example, 16 bits), it is easy a flash plate. For example, in the case of a 16-bit word, a run count

uses 163 bits. Therefore, the last word uses only a triplet. Thereby, in order to complete one word, it will be necessary to pad 13 bits. A skip count uses 85 bits. Therefore, only the 5 bits of the last words are not used. For this reason, in order to complete one word, it is necessary to pad 11 bits. Memory size is memory size >=(163+13) (/16)+ (85+11) (/16).

That is, if it is memory size >=17, independence is sufficient as padding of a run count and a skip count. If size is 16, one word contains a run count, a skip count, and PATINGU between both. An example of the 16-bit word equipped with both the run count and the skip count is shown in drawing 30 .

[0050] Drawing 2 B is the explanatory view of the memory for a variable-length run skip count. According to this memory structure, it is possible to start a run count from the one side (201) of memory, and to make a skip count start from the other side (202) of memory. Beginning or an end is sufficient as one side 201 of memory, and an end or beginning is sufficient as other side 202. Thereby, since the start of 1 run adjustable word and 1 skip adjustable word is required as it is simultaneously sudden, a run count and a skip count can be decrypted in concurrency. Therefore, it is not necessary to decrypt a run count first, and to judge the die length, next to decrypt a skip count. When it is serial-like, even

since [(since another side cannot be found unless one side is decrypted)] will become clear every (when it is the same as that of a run skip run and the following).

[0051] A separate skip count and a separate run count are used in order to decrypt information by the memory shown in drawing 2 B. Or one set of a decoder may be shared at both a skip count and a run count.

[0052] [Hardware for context models] A context model is realized by hardware. In hardware, one target is generating the following context as early as possible so that MQ encoder may not be in a standby condition.

[0053] [Memory configuration] (A) - (D) of drawing 3 is drawing showing the near multiplier and memory configuration for the example of a context model. The context model for the multipliers according to individual is based at most about on 3x3, as shown in drawing 3 A. In one example, 4 bits is processed simultaneously. In such a situation, the context model to the group of the multipliers 311-314 of four pieces is based at most about on 3x6, as shown in drawing 3 B. As for the memory access to hardware, it is often desirable to process the data by which grouping was carried out by the exponentiation of 2. Therefore, the fields based on the exponentiation of 2 which includes 3x6 fields

are 4x8 fields. (C) of drawing 3 is drawing showing about [which is the subset of 4x8 field of a multiplier] 3x6. Access to the whole 4x8 field of (C) of drawing 3 is performed as separate access of the same memory or memory ** which is not the same. (D) of drawing 3 shows 4x8 field divided into four 2x4 fields 301-304. 2x4 fields are stored in different memory according to an individual for the concurrent random access. in order to judge the contents from the group of a multiplier according to this memory structure -- all required information -- the sequence from memory -- not but, it becomes possible to read simultaneously. That is, 4x8 block of the whole of multiplier information is accessed simultaneously.

[0054] Drawing 4 is drawing showing one example of the significant memory configuration for random access to a 16x16 sign block. In the one example, although other sizes like 32x32 and 64x64 can be dealt with for example, it is not limited to such sizes. Each multiplier is assigned to one of four memory (A, B, C, or D) with reference to drawing 4 . Some groups (two lines of the top and two lines of the bottom) are the sizes of the one half other groups' size. The reason is to the first two lines of a sign block in (D) of drawing 3 from there being two lines of the top in the outside of a sign block (it having separated from the edge). The

same boundary condition appears also at the bottom of a sign block.

[0055] In one example, memory memorizes 1 bit (it is 8 bits at all about the single address) to the multiplier for significant condition. In other one example, memory remembers 2 bits (it is 16 bits at all about the single address) to be a multiplier for significant condition for every sign. In one still more nearly another example, such memory memorizes a total multiplier (it is 8Nbit when N expresses the size of the multiplier of one piece). In other one example, when all multipliers are not memorized by such memory, one auxiliary memory equipped with the one address for every multiplier is used.

[0056] It is one example changed into 7 bit-address output ("addr") to memory and 2-bit bank selection for a log (Verilog) code to specify [following] Memory A, B, and C or D for two 6 bit-address inputs ("x" and "y") from the control logic of a context (for 64x64 sign block) model very.

[0057]

module makeAddress(x, y, addr, bank); input [5:0] x; /* x has bits 5-0 and a bit 5 is MSB. */ input [5:0] y; output [6:0] addr; output [1:0] bank; wire [5:0] yp2; assign yp 2 = y+2; assign addr = {yp2 [5:3], x [5:2]} assign bank = {yp2 [2], x [1]} endmodule /* End.

[0058] The 1st assignment sentence sets up the offset over a boundary. That is, offset"assign yp2 = y+2" is used in order to perform suitable arrangement of the group of four lines, as shown in drawing 4 . The 2nd assignment sentence sets the address to the sum total of offset connected with the bits 5-3 of Input y, and the bits 5-2 of Input x as a bottom part of a number. The 3rd assignment sentence sets up the bank which is in agreement with the sum total of offset connected with the bit 2 of Input y, and the bit 1 of Input x.

[0059] [Significant propagation pass] Drawing 5 is drawing showing the memory used for the significant propagation pass for random access, and one example of a register. According to drawing 5 , Address A is inputted into Memory A in order to generate data output, and data output is held also at a register 501. Address B is answered, Memory B outputs data, and the outputted data are held also at the register register 502. Similarly, Memory C answers Address C, and outputs data, and an output is held at a register 503. Finally, Memory D answers Address D, and outputs data, and the data is memorized by the register 504. In one example, the outputs of each memory A-D are 2x4 fields, and create 4x8 fields (for example, field 601 of drawing 6) as a whole.

[0060] The full power of the memory shown in drawing 5 and a register gives 6x6

fields of a significant bit as a whole. It is necessary to notice this about the ability to become significant condition and a sign, or a actual multiplier in other examples. That is, the data used for juxtaposition from memory A-D are combined with the data which reading appearance was carried out to the front cycle from memory A-D, and were held at registers 501-504. The field with which this significant bit and the feedback from a context model were doubled is sufficient field in order to determine the pass with which 4x4 fields of a multiplier exist.

[0061] Drawing 6 is drawing showing the significant condition which reading appearance was carried out from memory and held at the register for random access. According to drawing 6 , a field 601 shows 4x8 fields by which reading appearance was carried out from memory. Reading appearance of the field 602 is carried out from memory A-D, and it shows 3x6 fields used for context modeling. A field 603 is memorized by registers 501-504, and shows 3x6 fields used for context modeling. A field 604 shows 4x4 fields of the multiplier processed. 2x4 8x8-block typical parts obtained from the memory location and register of memory A-D are shown in drawing 6 .

[0062] One example of the address-generation logic for significant propagation

pass is indicated by the following pseudocodes. It needs to be cautious of that addressing is not dependent on data, and zero data being prepared in a boundary. address_A_y=0 address_B_y=0 address_C_y=0 address_D_y=0 for y=0 to 60 step 4 address_A_x=0 address_C_x=0 read memory A (it registers with a degree)

read memory C (it registers with a degree)

assert clear for memory B register (next, cleared)

assert clear for memory D register (next, cleared)

for x=0 to 60 step 4 address_A_x=x +4 address_B_x=x address_C_x=x address_D_x=x if x < 60 then read memory A (it registers with a degree)

read memory C (it registers with a degree)

else use "all bits zero" for memory A output use "all bits zero" for memory B output read memory B (it registers with a degree)

read memory D (it registers with a degree)

4x4 blocks of multiplier x...x +3 and y...y +3 are processed. if y AND 4==0 address_A_y=address_A_y +8 address_B_y=address_B_y +8 else address_C_y=address_C_y +8 address_D_y=address_D_y +8 /* .

[0063] In order to process 4x4 blocks, the run of the bit of the same pass is dealt

with together. When the multiplier of N individual exists in the line of refinement MENTO pass, it is used in order that the following pseudocodes may process them.

[0064]

[0065] If the point that N is used instead of 1 is removed in order that this pseudocode may show that not the multiplier of one piece but the multiplier of N individual is processed, it is necessary to notice it about it being similar with the above-mentioned code.

[0066] When the multiplier of N individual exists in the line of clean-up pass, the following pseudocodes express one example of the process for processing a multiplier.

[0067]

[0068] Drawing 7 is the block diagram of one example of significant propagation pass logic. This logic is contained in bit modeling MQ encoder 29081-N of drawing 29 in one example. The pass for every multiplier is attached to per one multiplier, and is significant propagation or pass to as opposed to refinement MENTO or 4x4 fields which reached and were expressed by clean-up or the other triplet other than this other than this. In drawing 5 , by controlling access actuation of memory A-D, 4x4 blocks is taken out from memory, and significant propagation pass is performed. It is necessary to encode the multiplier under significant pass, and if four xeach 4 blocks is investigated, the run under much pass is identified, and on the other hand, a refinement MENTO- or as opposed to [reach and] clean-up pass run count and a skip count will be identified in order to process one run simultaneously. Within a block, a front (it depends in order of a scan) bit (or the case of the beginning of a new sign block pre- sign block) is a significant propagation bit, and, as for a current condition, not significant

propagation but a new run starts. In such a case, the increment of the index is carried out within the table holding a run skip count (for example, a skip is set to zero and a run is set to the 1st value). The table of both a run count and a skip count is carried out in this way, and the increment of it is carried out and it processes 4x4 blocks simultaneously. The bit in front of 4x4 blocks is in refinement MENTO pass or clean-up pass, and when many of such data follow, the increment of the count of a current run is carried out. When N expresses the exponentiation of 2, the field of other sizes including a 4xN field may be used.

[0069] The significant condition 701 for 8x8 fields for referring to drawing 7 is an input to the logic 702 which determines pass. Drawing 31 is the explanatory view of 8x8 typical fields. Significant condition 701 includes the information which shows that the multiplier of N individual exists in the line of for example, refinement MENTO pass. Such information is accessed from the above tables. The decision pass logic 702 investigates 16 fields [3x3] in the 6x6 field of the center of 8x8 fields. A coefficient A - I express the 1st 3x3 fields. Drawing 32 is the block diagram of one example of the decision pass logic 702. The logic in drawing 32 is repeated 16 times as every 1 time to each multiplier in 4x4 blocks. Sizes other than 3x3 field are sufficient as a field, and it is necessary to notice

the number of the fields processed about it not mattering at least even if [than 16 pieces] more [simultaneously].

[0070] With reference to drawing 32 , all the bits of a coefficient A - C are inputted into the OR gate 3201. A coefficient D and all the bits of F are supplied to the input of the OR gate 3202. All the bits of multiplier G-I are supplied to the input of the OR gate 3203. The output of the OR gates 3201-3203 is connected to the input of the OR gate 3204. The output of the OR gate 3204 is connected to the input of an inverter 3206, and the input of the AND gate 3208. A coefficient E expresses the 16-bit output of the refinement MENTO signal 704, the input of an inverter 3205 is supplied, and the output of an inverter 3205 is supplied to another input of the AND gate 3208, and the input of the AND gate 3207. The output of the AND gate 3208 is the significant propagation signal 703. The output of an inverter 3206 is supplied to other inputs of the AND gate 3207. The output of the AND gate 3207 is the clean-up signal 705.

[0071] Working, when either of the significant condition bits E is 0, it corresponds to the bit position of a bit whose output of the AND gate 3208 is 0, and when significant condition is 1 to either a coefficient A - D or multiplier F-I, the significant propagation signal 704 changes to 1. Similarly, when one bit of the

significant condition bits E is 0, the output of the AND gate 3207 corresponds to the bit position of the bit, and thereby, the clean-up signal 705 changes to 1, when all the significant condition bits to a coefficient A - D, or multiplier F-I are 0.

[0072] As a result of decision, logic 702 asserts the significant propagation signal 703, the refinement MENTO pass signal 704, or the clean-up pass signals 705 (in true condition). In one example, each signals 703-705 are 16-bit width of face. The only bit is 1 to each corresponding bit in signals 703, 704, and 705, and other two bits are 0. In this way, three kinds of probabilities exist in 16 locations. Each output of logic 702 is supplied to one input of the selection logic (for example, multiplexer (MUX)) 707.

[0073] The selection logic 707 generates three pass bits which show the pass to a current multiplier about a current multiplier, and sends a pass bit to the control logic 709. Only one of three pass bits answers the count signal 708 outputted from the control logic 709, and it is asserted. The count signal 708 shows whether which multiplier of the multipliers of 16 pieces in 4x4 blocks is a multiplier under current processing. When dealing with a refinement MENTO bit run and a clean-up bit run, the increment of the count signal 708 is carried out by the larger number than 1. In this way, the bit for every output of three pieces

corresponding to the multiplier is outputted among 16 bits of each output of the decision pass logic 702.

[0074] The refinement MENTO pass signal 704 and the clean-up pass signal 705 are inputted into a mask 705 with the feedback count signal 708. The count signal 708 is the current multiplier location in 4x4 field, 0...15 [for example,]. These inputs are answered, and a mask 705 carries out the mask of the already performed multiplier, and incorporates only the multiplier which is not yet encoded so that it may be shown by the count 708. For example, when the multiplier of three pieces is already processed, a mask 705 carries out the mask of each three signal line of a refinement MENTO output and a clean-up output (704 and 705).

[0075] Some (setting in the one example) signals generate the output expressing the signal 704 and signal 705 by which the mask was carried out to 1 of two pieces, and supply a mask 705 to the priority encoder 706. The output of a mask 705 is the refinement MENTO indicator and the clean-up indicator (for example, signal) by which the mask was carried out by which the mask was carried out.

[0076] According to two inputs, the priority encoder 706 detects the following

refinement MENTO bit (or multiplier) and the following non-clean-up bit to significant propagation pass, and inputs these bits into the control logic 709. In one example, the priority encoder 706 is a null detection priority encoder. The priority encoder 706 is changed into the count of the zero which precede the current position of the bit within a sign block (or multiplier) in that case. In one example, this is performed using the following table of truth value.

[0077] Table-of-truth-value input | Output 1 x x x x x | 00 1 x x x x | 10 1 1 x x x | 2 (it continues similarly hereafter).

[0078] A mask 705, the priority encoder 706, and the selection logic 707 receive the output from the decision pass unit 702, and constitute the processing unit which generates the output which shows the following non-refining MENTO multiplier and the following non-clean-up multiplier, and the pass to a current multiplier.

[0079] Answering an input, the control logic 709 generates refinement MENTO, the following index, a refinement MENTO run indicator, a refinement MENTO skip flag, clean-up and the following index, a clean-up run indicator, a clean-up skip flag, and a significant propagation indicator. The input to control logic is as follows.

[0080] the next non-refining bit position "R" -- the next non-clean-up bit position

"C"

[0081] The following pseudocodes explain the actuation of significant propagation pass logic shown in drawing 7 .

[0082]

count = 0 while (count < 16) mask = (1 << count)-1 refinement_masked = refinement | mask use-priority-encoder-to-find-next-non-refinement-bit

cleanup_mask = clean_up | mask use priority encoder-to-fine-next-non-cleanup bit if current bit is insignificance propagation passprocess coefficient as significance propagation count = count+1 else if current bit in refinement pass N= "next non-refinement bit". count process N bits as refinement pass count = count + N else N = "next non-cleanup bit" ? count process N bits as cleanup pass count = count+N /* .

[0083] Significant condition is updated from MQ decoder (under coding MQ encoder or a multiplier value) always, when "1" multiplier is encoded with significant propagation pass.

[0084] Considering the case where conte kiss TOMODE operates by one clock cycle, and MQ encoder operates by one clock cycle, two clock cycles are needed, when feedback exists. An example of the engine performance on 4x4 blocks from which the worst situation may happen to drawing 8 is shown. Eight context models and MQ encoder which operates like juxtaposition of a component clock rate by twice must be able to decrypt seven bit planes about one multiplier (8x2 / 2.25**7). When a skip does not arise within significant propagation pass, the engine performance of the worst situation falls up to 5.5 bit planes about per one multiplier at the maximum. the engine performance [in /

when a skip does not arise in which pass / the worst situation] -- per one multiplier -- at most -- it falls up to four bit planes.

[0085] [Significant propagation pass skip by software] In the case of software, the parallel access from much memory is unrealizable. Therefore, a sign block is divided into 4x4 groups of a multiplier in the one example. A count maintains the significant number of bits for every group. In such a case, the required amount of the maximum memory is 256x5 bits. The count of a block of all the multipliers that exist in refinement MENTO pass is 16. Clean-up is sufficient as all blocks of a count 0, and in order to investigate whether all are clean-up, they should just inspect near.

[0086] [Clean-up pass and refinement MENTO pass] In clean-up pass, addressing is generated using the following pseudocodes depending on data. The address x of the next multiplier in clean-up pass and y are inputted.

[0087]

module-cleanupAddress(x,y,addrA,addrB,addrC,addrD) input[5:0]x; input[5:0]y; output[6:0]addrA; output[6:0]addrB; output[6:0]addrC; output[6:0]addrD; wire[5:0]yp2; wire[4:0]ax; wire[4:0]bx; wire [4:0] cx; wire [4:0] dx; assign yp 2 = y+2; assign ax = (x[1:0] ==3) ?x[5:2]+1:x[5:2]; assign cx = (x[1:0]

==3) ?x[5:2]+1:x[5:2]; assign bx = (x[1:0] ==0) ?x[5:2]-1:x[5:2]; assign dx = (x[1:0] ==0) ?x[5:2]-1:x[5:2];assign ay = y[2] ?yp2[5:3]+1:yp2 [5:3]; assign by = y[2] ?yp2[5:3]+1:yp2 [5:3]; assign cy = yp2 [5:3]; assign dy = yp2 [5:3]; assign addrA = {ay, ax} assign addrB = {by, bx} assign addrC = {cy, cx} assign addrD = {dy, dx} endmodule /* .

[0088] Addressing used for clean-up pass is used also for refinement MENTO pass. However, in refinement MENTO pass, smaller near is also fully sufficient.

[0089]

If yp2[1:0] ==1 or (yp2[1:0] ==2) then if yp2 [2] == 1 then just read memories C and D else just read memories A and B else read memories A, B, C and D /* .

[0090] [Successive addressing to all pass] When performing successive addressing to all pass, an easy memory configuration is used [rather than] using two memory. Four xeach 4 field is assigned to one side of the two memory A and B with reference to drawing 9 . Thereby, all the parallel accesses demanded from 16x16 blocks become possible. As for the 1st sign block, only one half is used. When processing 8x8 blocks as shown in drawing 6 which offset is similar with above-mentioned offset, and does not hold the top data by which two lines is processed actually, it is because only the amount of [of a

multiplier] two lines correspond.

[0091] The memory of the memory way used for significant propagation pass and one example of a register are shown in drawing 10 . According to drawing 10 , Memory A answers Address A and generates data output. Similarly, Memory B answers Address B and generates data output. 2x2 crossbars 1003 have the input connected to the output of Memory A and B. One output of a crossbar is connected to a register 1001 and one output of a memory way. Another output of a crossbar is connected to a register 1002 and another output of a memory way. In this way, the output of Memory A and B is held at one of the registers 1001 and 1002, and is outputted to one of memory ways. The data by which reading appearance is carried out from Memory A and B are data to 4x4 fields. Registers 1001 and 1002 store 5x4 fields. When a register is loaded, 1x4 trains of most right-hand side are moved to 1x4 trains of most left-hand side, and other trains are put in from memory data output. A crossbar 1003 controls the output of the data to the suitable output of a memory way from Memory A and B by making data keep company with an output, when data of one line are processed at a time.

[0092] Drawing 11 is drawing explaining the situation used in order to prepare a

suitable field, since the memory and the register of drawing 10 are context model actuation. According to drawing 11 , fields 1102 are 4x4 fields of the multiplier which should be processed. A field 1101 expresses 5x6 fields (5x1 field of the bottom above 5x6 fields is disregarded) stored in the registers 1001 and 1002 used for context modeling. Fields 1103 are 4x8 fields from memory. Fields 1104 are 1x6 fields from the memory used for context modeling.

[0093] One example of the pseudocode for memory addressing to all the three coding pass is explained below.

[0094]

address_A_y=0 address_B_y=0 for y=0 to 60 step 4 address_A_x=0 address_B_x=0 clear registers read memory A (memory A which will be registered into the degree)

read memory B (memory B which will be registered into the degree)

for x=0 to 60 step 4 address_A_x=x +4 address_B_x=x +4 if x < 60 then read memory A (memory A which will be registered into the degree)

read memory B (memory B which will be registered into the degree)

else use "all bits zero" for memory A output use "all bits zero" for memory B output process 4x4 block of-coefficients-x...x+3,y...y+3 if y AND 4==0

address_A_y=address_A_y+8 elseaddress_B_y=address_B_y+8 memory holds a condition in order to show the right pass for refinement MENTO pass and clean-up pass. A condition is 2 bits per multiplier in order to identify three conditions (significant propagation, clean-up, and refinement MENTO).

[0095] During significant propagation pass, about all the multipliers of 16 pieces, in the case of a significant multiplier, a condition is set up by refinement MENTO, and, in the case of an un-significant multiplier, is set up by juxtaposition at clean-up. While the processing about the multiplier of 16 pieces continues, the condition of the multiplier which exists in significant propagation pass is changed into significant propagation from clean-up. For every multiplier, a condition is 1 bit and calls this 1 bit a pass bit. In one example, significant condition and a pass bit are used in order to determine right pass. A table 6 illustrates the direction for use of a pass bit. Since 1 bit is used about one multiplier, memory usage decreases rather than the run skip counting method explained here.

[0096]

[A table 6]

In one example, * in refinement MENTO pass of a table 6 is performed to juxtaposition about all 16 multipliers at the time of the start of processing to 4x4 blocks.

[0097] The 48-bit width of face equipped with significant condition, the pass bit, and the sign binary digit for every multiplier is sufficient as the memory which gives access to 2x4 fields.

[0098] Drawing 12 is the block diagram of one example of the pass decision logic which uses a priority encoder in order to detect each multiplier which realizes the above-mentioned table 6 and exists in the present pass. With reference to drawing 12 , the decision pass logic 1203 receives the significant condition 1201 over 6x6 fields, the pass bit 1202 to 4x4 fields, and the present

pass signal (or other indicators) 1220 that shows the present pass. The pass bit 1202 includes the signal (namely, 16 signals) for every multiplier in 4x4 field. These inputs are answered, and the decision pass logic 1203 generates an output in order to specify the pass to 4x4 fields. In that case, the decision pass logic 1203 asserts the signal 1204 which shows the significant propagation pass bit in significant propagation pass, the signal 1205 which shows the refinement MENTO pass bit to the multiplier in refinement MENTO pass, or the signal 1206 which shows the clean-up pass to the multiplier in clean-up pass to each multiplier in 4x4 fields.

[0099] The selection logic 1207 answers the current pass signal 1220, and outputs one of indicators 1204-1206 to the mask logic 1208. In one example, the selection logic 1207 contains 16x3:1 multiplexers (MUX). The mask logic 1208 answers the count signal 12010, and generates a signal, and a count signal specifies the multiplier by which current processing is carried out. The output of a mask 1208 is supplied to the priority encoder 1209, and the priority encoder 1209 outputs the signal to the control logic 1212. The mask logic 1208 and the priority encoder 1209 operate like the logic of the same identifier and the encoder which were shown in drawing 7 . The control logic 1212 answers an

input, generates a sign or an idle status indicator to a signal line 1213, and generates the count signal 1210.

[0100] The following pass bit logic 1211 receives the output (the location under current processing is shown) from the priority encoder 1209, the current pass signal 1220, the new significant condition 1221 from MQ encoder, and the refinement MENTO pass signal 1205. Significant condition information is expressed by showing whether the refinement MENTO pass signal 1205 has the significant multiplier of precedence. The current pass signal 1220 and the new significant condition 1221 show whether processing is performed by clean-up pass as a whole. This input is answered, the following pass logic 1211 generates the following pass bit, and this following pass bit is used as an output, in order to distinguish the case of a sign 0 and the case of a sign 1 in a table 6. The following pass bit is held at memory, next is used as a pass bit 1202.

[0101] Next, a pseudocode explains actuation of the logic in drawing 12 . Such a function is included in MQ encoder 29081-N. Significant condition and a pass bit are cleared before processing the 1st clean-up pass.

[0102]

count=0 if significance-propagation pass then set-next-pass-bit-to "1" for all

coefficients-in-refinement pass while (count<16) if-significance-propagation-pass-then in_pass=coefficients in significance propagation-pass else if refinement pass then in_pass=coefficients in refinement pass else in_pass=coefficients in cleanup pass mask=(1<<count)-1 in_pass_masked=in_pass AND (NOT mask) use priority encoder to find next coefficients in pass, N if next coefficient found code coeff N if significance propagation pass then next pass bit=NOT next significance state else nextpass bit=pass bit count=N +1 else count=16 idle for coeff N next pass bit=pass bit /* .

[0103] In the above-mentioned code, variable in_pass is the output of a 3:1 multiplexing function. Variable mask expresses a mask and variable in_pass_masked expresses the result of having applied the mask. Variable N expresses the next multiplier under pass, and is the output of a priority encoder. Detection of Variable N continues the control function of a coding scheme after that.

[0104] The above-mentioned code code coeff N means encoding a multiplier, when a multiplier exists in current pass. Code idle for coeff N is performed while processing a run.

[0105] [Double context generation] A context often produces a serious feedback

loop in time between MQ encoder and a context model depending on the bit encoded at the end in the case of a decryption. Since this context model delay is shortened to the time amount for simple multiplexing, for a reason in case one context of a sake in case the present bit under decryption is 0, and the present bit under decryption are 1, while will accept a context model, and it can generate two contexts of a context. Selection of a context is performed when a bit is known.

[0106] Drawing 13 is the block diagram of one example of double context generation logic. With reference to drawing 13 , a context model generates a context 0, enabling [0], a context 1, and enabling [1]. A context model generates two contexts 0 and 1, and both contexts are sent to a multiplexer (MUX) 1302. It connects so that a signal may be received, and a multiplexer 1302 generates the context indicator in which it is shown whether a context is effective, and the enabling indicator in which it is shown whether the bit should be encoded or not. These outputs are supplied to the input of the MQ encoder 1303, and MQ encoder generates a bit. The output bit from the MQ encoder 1303 is used by the multiplexer 1302 in order to choose the context which should be outputted to the MQ encoder 1303. In this way, a context model generates a

context when a current bit is decrypted as 0, and two contexts of another context when a current bit is encoded as 1, and the output bit for the MQ encoder 1303 chooses the context of the right direction.

[0107] To the run length coding for clean-up pass, a table 7 shows the following two contexts which may happen for every case, and follows the flow of the clean-up pass indicated by JPEG 2000 criterion. The value of the bit encoded in the run length context is used in order to judge whether the following context is an object for the groups of the four following multipliers, or it is the context of the homogeneity for the groups of four current multipliers. When this bit is 0, the run length coding which makes MQ encoder an idle state following it for a next cycle in case four multipliers are expressed by one judgment does not have a serious adverse effect for the rate engine performance. After the bit encoded in the 2nd homogeneity context (uniform B) is a sign binary digit always directly encoded in the context 9 (XOR bit 0) in JPEG 2000 criterion. Here, when an XOR bit is 0, it means that a sign is not reversed.

[0108]

[A table 7]

A and B in a table 7 express two bits indicated by "the following 2 bits (the next two bits, returned with the UNIFORM context) returned with a uniform context" into the section D3.4 of JPEG 2000 criterion.

[0109] The example which does not use run length coding in significant propagation and clean-up coding pass is shown in a table 8. Although a magnitude bit is encoded, a current multiplier is generated to the magnitude of the following multiplier, assuming that it is 0, or, as for a context, a sign binary digit context is generated to a current multiplier.

[0110]

[A table 8]

In refinement MENTO pass, the refinement MENTO multiplier encoded previously does not affect a context.

[0111] [MQ encoder]

MQ decoder data flow diagram 14B accompanied by a rate context is the block diagram of the typical example of decryption implementation. In drawing 14 B, the context model 1430 supplies a context to memory 1431, and a probability condition is determined by memory. A probability condition is changed into Qe value for the algebraic-sign machine 1433 using logic 1432, and the algebraic-sign machine 1433 updates an internal A&C register, and determines a judgment result (MPS or LPS). These are all performed, before being able to determine the following context typically. In the case of almost all the example of hardware implementation, a decryption rate is restricted by the large (it feeds back to the context model 1431) feedback loop.

[0112] On the other hand, drawing 14 A is the block diagram of one example of an initial context MQ decoder. In this example, a feedback loop possesses the very easy logic 1407 instead of the whole decryption actuation. Therefore, almost all decode and updating may be performed in juxtaposition using the low order feedback loop 1401.

[0113] With reference to drawing 14 A, the sign stream 1400 is inputted and an internal state 1401 is updated. In one example, A of an internal state and C register specify a current interval as indicated by the Appendix C of JPEG 2000 criterion. Register A specifies a current interval and the sign register C is connection of Chigh and a Clow register.

[0114] A context 1402 is given with the context model 1410. It is used in order that a context 1402 may investigate the probability condition 1404 of memory 1403, and a probability condition is changed into the probability class (Qe value) 1406 by logic 1405. The Qe value 1406 expresses the current estimate of a low probability symbol (LPS). The Qe value 1406 is compared with A register value and C register value which are indicated in drawing C-15 of JPEG 2000 criterion of the internal state of MQ encoder in order to generate the output judging result 1408 using logic 1407. An output judging result is a high probability symbol (MPS) with a more high probability, or LPS. The output judging result 1408 is inputted into the context model 1410. In one example, the operation about Qe value and an internal state needs 16 bit operations. The operation of these blocks decrypts a judgment result which is indicated into the section C.3.2 of JPEG 2000 criterion.

[0115] Drawing 15 is the block diagram of one example of an anaphase context MQ decoder. In drawing 15 , 16-bit processing is removed from the context model-feedback loop formation. It is received as an input to the updating logic 1503, and updating logic updates an internal state and the sign stream 1601 contains A register and C register which specify a current interval. A sign stream is inputted into new A register value and new C register value, and a list to logic 1504, and logic 1504 generates two p classes 1509 which are explained below (pclass), i.e., p class, and the p class 1510. Two p classes are inputted into the comparison logic 1511 and 1512.

[0116] The context model 1520 generates a context 1502. A context 1502 is used in order to investigate the probability condition 1506 of memory 1505. In one example, memory 1505 contains a look-up table. Qe value can be determined now by checking the probability condition 1506. The probability condition 1506 outputted from memory 1505 is changed into the probability class (index) 1508 by logic 1507.

[0117] The comparison logic 1511 judges whether the p class 1509 is larger than the probability class index 1508, and it compares in order to judge whether the comparison logic 1512 has the probability class index 1508 larger than the p

class 1510. When both comparison results are truth, the result of both comparison logic 1511 and 1512 is inputted into the AND gate 1513 so that a judgment result may be outputted. This judgment result is MPS or LPS. In this way, a context 1502 is changed into the 5-bit probability class index 1508 (since the value of 32 pieces can be taken to Qe value in JPEG 2000). An internal state is used in order to generate two 5-bit probability class indexes. When the index corresponding to a context is in the outside of two indexes generated from the condition, a judgment result is MPS, otherwise a judgment result is LPS (that is, it is inside two indexes).

[0118] The important advantage of the example of drawing 15 does not have the successive renewal of an internal state, as shown in drawing 14 B, and it is carried out in parallel to generation of the following probability class (index) 1508. Moreover, since, as for two probability classes, only 5 bits is measured with p class index, an operation is simplified dramatically.

[0119] The logic 1504 of drawing 15 creates the information shown in drawing 16 A. If a value is given to A register and B register, logic 1504 will judge whether it determines what two dividing points over p class are, next a sign is between dividing points, or it is outside. This may be performed to juxtaposition.

[0120] Drawing 16 A is drawing showing how the comparison of a probability class index is performed. In drawing 16 A, the p class 0 is a case in which almost all intervals are turned to MPS and which was distorted dramatically. To the p class 1 to the p class 4, this distortion becomes small and an MPS interval is reduced. conditional [which produces the p class 5 to a neighboring probability 50% in MPS] -- exchange is given. To some probability classes, a known condition is MPS and has the sign stream value (sign) which is LPS to other probability classes. Since the probability class can be set in order, in order to judge whether a sign is MPS or it is LPS, just two comparisons are enough. That is, in drawing 16 A, when the location of a sign is given in the state of known, a judgment result is MPS to the p classes 0-3, always LPS to the p class 4, and MPS also to the p class 5. What is necessary is to judge only two break points (the break point between the p class 3 and the p class 4, and break point between the p class 4 and the p class 5) rather than to find whether it is MPS or it is LPS to each probability class. Therefore, when QE value is given, it is determined as the space where a break point exists what the probability class which appears actually is (when a probability class / index is known).

[0121] The same approach in hardware is used in order to judge MPS or LPS for

every Qe value which may happen, next to multiplex the result. For example, the multiplexer 1610 which has much inputs is shown, and each input is related with p class by drawing 16 B, and produces either MPS or LPS as an output in it according to a sign.

[0122] [Multibit decryption by MQ encoder] Much MPS can be simultaneously decrypted, unless MPS which needs normalization exists, or as long as only the last MPS needs normalization (since the same P class is continued and used). Drawing 17 shows the interval for a multiplex MPS decryption. Standardly, if the location of the sign stream to the interval specified with A register and C register and a difference with Qe value are two or more, it is possible to encode much MPS. When interval size is divided by Qe value and a decoder stops at the same context (i.e., when stopping at the same probability class), it is possible to decrypt much MPS simultaneously. For example, when a sign stream is investigated and it turns out that being processed is 16 bits, the location of the sign stream in the interval specified with A register and C register estranges only the multiple of Qe value, being due to be used in order to process the same context, i.e., the data whose same probability classes are many cycles, is shown, and much MPS is decrypted simultaneously. That is, when {(interval specified

with A register and B register) - [-} (location of a sign stream)]/Qe is judged and it is rounded off by the minimum larger integer than this value, that result expresses the number of MPS decrypted simultaneously. This count is performed by well-known hardware.

[0123] [Typical implementation gestalt of 5 and 3 filter] Set in the one example. Reversible and irreversible [5], and 3 wavelet filter are used. Here, 5 and 3 are the numbers of taps in a wavelet filter, i.e., the number of the non-zero (continuation) values in the basis function support of a kernel. Reversible means that the same number is strictly obtained with an output with an input, if rectification and inverse transformation are performed. About input precision, since the increment in the precision which can be predicted [that it is moderate and] is a medium mathematical term, it is required. That is, regular distortion is not introduced by mathematical precision. Irreversible means that a very high precision is required, in order to guarantee that there is no mathematical distortion (exact playback). However, a actual top and irreversible filter is combined with the quantization which produces distortion which overwhelms regular mathematics precision distortion in almost all cases.

[0124] One example of a rectification filter is shown in drawing 24 . According to

drawing 24 , line x0x1 and last x0 are supplied to the high pass (highpass) filter 2402 from a line buffer 2401 to one input of reception and the low pass (low-pass) filter 2404, and it generates the output held at a line buffer 2403. Line buffers 2401 and 2403 hold one line which has tile width of face. A low pass filter 2404 generates reception and an output for the output of the cycle in front of a high-pass filter 2402, i.e., the output from a line buffer 2403, with current x0 line. The output of the low pass filter 2404 to two past clock cycles is delayed with the delay vessels 2405 and 2406, and gives the filter output before 1 cycle, and the filter output before a two cycle.

[0125] The output of the past of a high-pass filter 2402 is delayed with the delay vessels 2407 and 2408, and the current output of a high-pass filter 2402 and the last output of two pieces of a high-pass filter 2402 are supplied to a high-pass filter 2413. The output of a high-pass filter 2413 is a multiplier in HH subband, and is supplied to a low pass filter 2415 with the output of the past in front of the two cycle of a high-pass filter 2402, i.e., the output from the delay machine 2408, and the output before a high-pass filter 2413. The output of a low pass filter 2415 is an output from HL subband.

[0126] The output of a low pass filter 2404 is supplied to a high-pass filter 2409

with the output of the delay machines 2405 and 2406. The output of a high-pass filter 2409 is LH subband.

[0127] The output of a high-pass filter 2409 is supplied to the input of a low pass filter 2411 with the output of the delay machine 2406, and the output before the high-pass filter 2409 delayed with the delay vessel 2410. The output of a low pass filter 2411 is LL subband. When LL subband which is the output of a low pass filter 2411 is supplied to a line buffer 2412, the output of a line buffer 2412 expresses the input to the next level of wavelet transform with the output of a low pass filter 2411. The next level of wavelet transform carries out cascade connection of the wavelet transform shown in drawing 24 .

[0128] Drawing 25 A is the explanatory view of one example of a low pass filter which is used by the conversion (for example, above-mentioned 5, 3 conversion) explained here. A low pass filter is designed so that an output may be generated based on the function according to -x0+2x1-x2. In the case of invertible transformation, a low pass filter is a degree type [0129].

[Equation 1]

It is alike, and follows and operates.

[0130] If drawing 25 A is referred to, an adder 2501 will add x0 line of the last with the present x0 line. A least significant bit output expresses the output of the high-pass filter of drawing 25 B, and is used for irreversible conversion. The remaining bits are supplied to a subtractor 2502, and they are subtracted from x1 input in order to generate the output expressing the most significant bit. In the case of invertible transformation, all of these most significant bits are required. A subtractor 2502 is permuted by the adder in order to change the filter of drawing 25 A to the reverse wavelet filter for using it as an odd number (highpass) filter by inverse transformation in the case of reverse wavelet transform. Such an example is shown in the high-pass filter of drawing 25 B.

[0131] Drawing 26 A is the explanatory view of one example of the high-pass filter used for the conversion explained here. In irreversible conversion, a high-pass filter operates according to the following formulas and 4x1-x0-x2.

[0132] In the case of invertible transformation, a high-pass filter is a degree type and [0133].

[Equation 2]

It is alike, and follows and operates.

[0134] If drawing 26 A is referred to, an adder 2601 will add either the reversible side of x0 line of the last, or an irreversible side to the present x0 line. The output of an adder 2601 is rounded off using an adder 2603, and is added to a term. In by the side of reversible, a rounding-off term is 2, in by the side of irreversible, is 0 and is supplied by the multiplexer 2602. Using an adder 2604, the output except 2 bits of low order of an adder 2603 is added to x1 line, and produces a reversible side output. 2 bits of low order of the output of an adder 2603 and the output of an adder 2604 express an irreversible side output.

[0135] An easy change can be realized without [in order to change a reversible and irreversible side by using a multiplexer 2602, without it needs the completely separate hardware for both functions, or] requiring that reversible side rounding-off affects an irreversible side output.

[0136] In the case of reverse wavelet transform, an adder 2604 is permuted by the subtractor in order to change the filter of drawing 26 A to the reverse wavelet filter for using it as even number under inverse transformation (low pass filter). Such an example is shown in the low pass filter of drawing 26 B.

[0137] Drawing 27 is drawing showing the alternative-example of the conversion in drawing 24 , and contains the multiplexer for performing mirror-image processing on an image boundary. A multiplexer is constituted by multiplexers 2701-2712. For example, a multiplexer 2701 uses x0 line instead of x0 line of the last on a boundary, when a line does not exist in a buffer 2401 (for example, top of a tile). A multiplexer 2702 enables it to supply the input of others [line buffer] to a low pass filter 2404, when further x0 line which should be inputted when it arrives at the bottom of a tile does not exist. When a line does not exist in a buffer 2403, the output of a high-pass filter 2402 can make it enable it similarly to use a multiplexer 2703 as an input to a low pass filter 2402. A multiplexer 2704 enables it to supply the input to a low pass filter 2404 from a line buffer 2403, when there is no output from a high-pass filter 2402. Multiplexers 2705 and 2706 enable it to supply the output of the delay machine 2406, and the output of a low pass filter 2404 to the input to a high-pass filter 2409, respectively, when the output to a low pass filter 2404 and the output from the delay machine 2406 cannot be used. It is realized also about multiplexers 2709 and 2701, multiplexers 2707 and 2708, and multiplexers 2711 and 2712 that it is the same as that of multiplexers 2705 and 2706.

[0138] Drawing 28 is the block diagram of one example of reverse 5 and 3 conversion. With reference to drawing 28 , the even number filter 2815 is connected so that LL multiplier, HL multiplier, and HL multiplier of the cycle of the front from the delay machine 28801 may be received. the output of the even number filter 2815 -- the even number filter 2811 -- much more -- the force and the delay machine 2802 -- much more -- the force and the odd number filter 2803 -- it connects with the force much more. HL multiplier of a front cycle is supplied to another input of the odd number filter 2803 through the delay machine 2801, and the output of the cycle in front of the even number filter 2815 is supplied to it through the delay machine 2802. The output of the odd number filter 2803 is connected to one input of the even number filter 2810.

[0139] Since the filter 2805 is connected so that current HH multiplier and LH multiplier, and HH multiplier in the cycle of the front from the delay machine 2804 may be received, the above-mentioned configuration and the same configuration exist also about LH multiplier and HH multiplier. The output of the even number filter 2805 is connected to one input of the even number filter 2811, the input of the delay machine 2806, and one input of the odd number filter 2807. HL multiplier (namely, output of the delay machine 2804) of a front cycle, the output

(namely, output of the delay machine 2806) of the cycle in front of the even number filter 2805, and ** are supplied to another input of the odd number filter 2807. The output of the odd number filter 2807 is connected to one input of the even number filter 2810.

[0140] The output of the even number filter 2805 and the output of the odd number filter 2807 are supplied to a line buffer 2808 and a line buffer 2809, and it is necessary to notice them about being held, respectively. The size of line buffers 2808 and 2809 is in agreement with one half of tile width of face. The output of a line buffer 2808 is connected to another input of the even number filter 2811, and one input of the odd number filter 2815. The output of a line buffer 2808 is connected to one input of the even number filter 2810, and one input of the odd number filter 2814.

[0141] The output of the even number filter 2810 is the "C" part of the image data outputted, is held at a line buffer 2812 and supplied to the input of the odd number filter 2814. In one example, the size of a line buffer 2812 is in agreement with one fourth of tile width of face. Answering this input, the odd number filter 2814 generates the data corresponding to the "A" part of image data.

[0142] The output of the even number filter 2811 corresponds to the "D" part of

image data, is supplied to one input of the odd number filter 2815, and is held at a line buffer 2813. In one example, the size of a line buffer 2813 is 1/4 of tile width of face. The output of a line buffer 2813 is connected to another input of the odd number filter 2815. The output of the odd number filter 2815 corresponds to the "B" part of image data.

[0143] [Other parallel system operation techniques]

In the case of the example of the allotment hardware to the encoder for juxtaposition of a sign block, it is effective in juxtaposition within the same tile to encode much sign blocks. Drawing 21 is the memory usage explanatory view of one example of the encoder containing many MQ encoders. Each MQ encoder possesses the related context model, and it is used in order to process much sign blocks.

[0144] As for each MQ encoder, reference of drawing 21 assigns memory (for example, separate memory, a single, or the part of much memory). In one example, the part with the assigned memory holds coded data, and die length, a zero bit plane, and coding pass are held at another part of memory.

[0145] Allocation to the juxtaposition unit of 128x128 tiles, a 64x64 sign block, and the sign block about each conversion level is shown in drawing 18 -20. The

amount of coding by which allocation is performed for every juxtaposition encoder is performed for being balanced. In one example, allocation of a sign block is performed so that each MQ encoder is the most possible possible range, and the multiplier of the abbreviation same number may be encoded, maintaining balance between an upper level multiplier and a lower level multiplier. The configuration of those other than this is also realizable.

[0146] The example of allocation of the sign block about 4:4:4 data when four sets, six sets, and eight sets of MQ encoders are used for juxtaposition is shown in drawing 18 A, B, and C, respectively. In drawing 18 C, since the sign block assigned to juxtaposition unit"H" (1HH chrominance subband) is often heavily quantized when eight units are parallel, this unit has high possibility that the number of the multipliers which can be processed to per unit time amount will increase more than other units (since the non-zero bit plane which should be encoded hardly exists).

[0147] The example of allocation of the sign block about 4:2:2 data when four sets, six sets, and eight sets of MQ encoders are used for juxtaposition is shown in drawing 19 A, B, and C, respectively.

[0148] The example of allocation of the sign block about 4:1:1 data when four

sets, six sets, and eight sets of MQ encoders are used for juxtaposition is shown in drawing 20 A, B, and C, respectively. In drawing 20 C, when eight units are parallel, it is expected that the number of the multipliers which Units C, D, and E can process to per unit time amount increases more than other units.

[0149] The encoder of drawing 29 is used in order to perform above-mentioned coding. For example, each encoder of the MQ encoders of N base of bit modeling MQ encoder 29081-N is assigned to either of A-H shown in drawing 18 thru/or 20.

[0150] Although even sets of MQ encoders are made juxtaposition about drawing 18 thru/or 20, odd sets are sufficient as the number of MQ encoder made juxtaposition.

[0151] When it is not memory economization the decryption which is not lost for storing a multiplier in hardware, in order to reduce the memory usage holding a multiplier, it is possible to use zero bit plane conversion. When hardware can hold the bit plane of N individual for every multiplier, a decryption is completed after the bit plane of N individual is decrypted. A consecutive bit plane can be quantized (rounding-off processing).

[0152] The example to which drawing 22 A uses the bit plane of the finite

individual of memory for every multiplier during coding is shown. For example, eight bit planes (N= 8) of memory are standard transcriptions, and since a multiplier is encoded by 16 bits, they may be used. These multipliers are some subbands other than LL (LL subband multiplier is not quantized) subband, and are generated as a result of having applied wavelet transform to image data. In one example, wavelet transform contains 5 and 3 wavelet transforms. 5 of a large number which carry out parallel operation, and 3 wavelet transforms may constitute wavelet transform in order to generate LL subband, HH subband, LH subband, and HL subband to juxtaposition. The memory holding the multiplier from wavelet transform is accessed with a context model in order to perform coding based on a multiplier bit.

[0153] During coding, a multiplier is saved, before the number of zero bit planes becomes a base. A counter counts the number of the initial zero to the bit planes 8-15 of a high order. As long as bit planes 8-15 are zero altogether, memory holds the information (magnitude) over the corresponding bit planes 0-7. When 1 appears in bit planes 8-15, the corresponding counter stops and memory holds the information on the corresponding bit planes 8-15. When, as for the bit planes 0-7 with which the counter specified zero altogether and corresponded to the last

of coding of a sign block to bit planes 8-15, a counter stores a value in the last of memory, it holds in memory, or a counter needs to specify the starting address to the bit planes 8-15 in memory, and needs to carry out rounding-off processing (quantization) of the corresponding bit planes 0-7. In this way, a count acts as side band information and shows that the information held at the memory array to the location in the line specified by the count from the beginning of a line became data without the need.

[0154] The separate bit plane of memory is used in order to hold sign information, or sign information is held with significant condition.

[0155] In other examples, the memory of a small amount is used instead of a counter for (variable-length VL) sign information (for example, run length sign). It enables this to store the bit plane containing a small number of "1" bit in a part of memory for every bit plane. After storing a bit in memory, a context model accesses memory in order to acquire a bit. However, since each line contains potentially the data which should be quantized, it does not have the need of being accessed and used with a context model. Drawing 22 B is the block diagram of one example of the control logic for controlling access to memory. This logic collaborates with the context model which accesses memory, or

operates as some context models.

[0156] With reference to drawing 22 B, Address addr accesses the memory array 2201 which generates a bit. The counter value relevant to the line of memory which held the address and the address is supplied to the comparison logic 2210. In the comparison logic 2210, when it judges that the address is beyond a counter value over a line, the 1-bit output from a memory array 2201 is generated, otherwise, zero are outputted.

[0157] Drawing 23 is drawing showing a part of memory from VL sign, and the memory array which stores a multiplier. VL sign is used in order to specify 1-bit existence by showing the number of bits skipped until the following "1" appears within a line. In this way, this VL sign is created in order to specify two counts that access logic can know the location of the next bit plane. Other VL signs are used in order to give three or more counts. By using VL sign, one full twist can also use the bit plane of little memory now typically. If the size of small capacity memory is 1/32 of the size of a sign (per one bit plane) block, R2 [8] sign or R2 [7] sign may be used. R2 [8], R2 [6], and the detailed information on R2 [7] sign are indicated by U.S. Pat. No. 5,381,145 and name "Method and Apparatus for Parallel Decoding and Encoding of Data" of invention by which inheritance was

carried out to the assignee of this application, and January 10, 1995 issuance.

[0158] When conversion and juxtaposition actuation of a context model / MQ encoder are desirable videos, two memory banks are needed. In the case of static-image application, one memory bank is enough because of conversion and successive actuation of a context model / MQ encoder.

[0159] Although the above-mentioned memory economization technique is explained about the line, the memory area of arbitration, such as a train, a block, a page, and a field, can be used, for example. Or separate memory may be used.

[0160] A packet header is created in order to create packet header processing, for example, a sign stream like a JPEG 2000 sign stream (or bit stream). In one example, this information is accompanied by the tag tree structure in order to deal with the sign block of the number of arbitration. In a certain kind of situation, the tile header equipped with the sign block of the restricted number for tiles is created. For example, the number of the sign block with which a tile is encoded as one when each subband is divided into a 64x64 sign block including four 128x128 subbands is four pieces. A packet header specifies the number of a zero bit plane in case data exist [whether the data to a specific sign block exist, and], the die length of coded data, and the number of the pass for coding

included in data.

[0161] The following table 9 shows one example of the packet structure containing a 2x2 sign block and one layer for packets. With reference to a table 9, the tag tree structure has only the depth of 2 level. The location which notation"z" showed the location of the tag tree structure information on the zero bit plane of a record level, and was shown by notation"_" shows the location on which the remaining zero bit planes, the pass for coding, and die-length information are put.

[0162]

[A table 9]

* expresses 0, 10, or 110000 among a table.

[0163] In one example, for convenience, a sign 110000 is used, when [of the example of implementation] a sign block is not included.

[0164] The initialization with one following example of the procedure which writes in the packet header equipped with a restricted number of sign blocks and single layers for tiles: Begin maximum from setting out in the zero bit plane minimum value MZP for every subband.

[0165] In one example, the case to 15 bit planes, the maximum of MZP is 0xF and is 0x1F the case to 31 bit planes. While the value used becomes large, the number of the bit planes which can be dealt with in the example increases.

[0166] Next, when encoding the multiplier of each sign block within a packet, it is

Save included or not bit (the bit which is not included or included is saved).

Save number of zero bitplanes (the number of a zero bit plane is saved)

If zero bitplanes less than MZP then MZP = zero bitplanee (if a zero bit plane is under MZP, it is a MZP= zero bit plane)

Save number of coding passes (the number of the pass for coding is saved)

Save Length (die length is saved)

It performs.

[0167] When all multipliers (quantization back) are zero, Save included or not bit is set and specifies that a sign block is not included. Eventually, after the information on a tile or a subband is processed, a packet header is described as

follows.

[0168]

write "1" for each-subband write "1" first_flag = 1 for-each-code-block if-not-included-then write "0" else write "1" if-first_flag-then write MZP in tag tree format first_flag = 0 write zero bitplanes ?MZP in tag tree format write codingpasses determine minimum Lblock value write Lblock write length -- here, Lblock is specified into the section B.10.7.1 of JPEG 2000 criterion.

[0169] A packet header is at least 1 byte, and it is necessary to notice a JPEG 2000 conformity type decoder about the ability of the written-in information to be recognized.

[0170] When many layers exist, initialization of a MZP variable is performed like the cutting tool of one layer. During coding of each sign block, the index of inclusion or not including, the number of the pass for coding, and die length are saved for every layer. Furthermore, the following initialization processings are desirable.

[0171] In first_flag = 1 initialize Lblock for each code-blockinitialize already included for each code-block to false 1 example, Lblock is initialized 3. When "already included" is truth, it means that some precedence layers encoded data

(that is, the sign block has appeared in the past).

[0172] The following procedures are used in order to write in a packet for every layer.

[0173]

write "1" for each-subband if layer 0 then write "1" for-each-code-block if-not-included-then write "0" else write "1" if-code-block-not-already-included-then if first_flag then write MZP in tag tree format first_flag = 0 write zero bitplanes. MZP in tag tree format set already included write coding passes determine minimum Lblock value write Lblock A separate bit is sufficient as write length"already included" information to each sign block. Or the value for which a zero bit plane is not used may be used in order to specify "already included". For example, when 14 bit planes exist, it is possible by setting a zero bit plane to 15 (0xF) for "already included" to be shown.

[0174] In the case of compression coded data JPEG 2000 which do not use "0" packets, a packet header is rounded off per cutting tool. In some cases, a packet header stores the bit of a large number smaller than the number of bits required in order to hold only 0 bits or an only packet header in the byte boundary. Generally a packet header is rounded off by the cutting tool by padding.

Moreover, in the one example, although an activity of the transcription that a packet header transcription is not peculiar and shortest typical possible is desired, if instead of [of the bit location where the superfluous bit used will be buried by padding] is carried out, the transcription which is not the shortest format considered will be used. This is especially effective when the information encoded in the superfluous bit shows the information about the following packet in tile component level division.

[0175] For example, a zero packet is outputted, when a single subband exists and 2x2 blocks is included. However, in the same amount of space, something exists in a packet, and in order to show that nothing is included by the level of the top of the tag tree structure, zero may be outputted. Or it is able for something to exist in the tag tree structure and to show what is been 0000 (that is, nothing exists in four places according to an individual). In this way, these superfluous bits are used in order to give more tag tree structure information. This tag tree structure information is the information which appears in a packet header and will surely be advanced later. By advancing a bit to the packet header of precedence, it is possible to reduce size of the whole sign stream 1 byte at a time (or more than it).

[0176] An alternative and modification of the above explanation to a majority of this inventions will become clear to this contractor. However, it explains for instantiation and it is necessary to notice the illustrated concrete example about not having the intention of restricting this invention. Therefore, it is not what meant restricting the range of the matter mentioned to the claim only the matter considered that explanation of the detail of many examples is indispensable to this invention was indicated to be.

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]

[Drawing 1] It is the block diagram of a JPEG2000 decryption system.

[Drawing 2 A] It is drawing showing an example of the 8x8 sign block of a multiplier with which the label in which subbit plane pass is identified for every multiplier, and the sequence of processing for each coding pass is shown was attached.

[Drawing 2 B] It is the explanatory view of the memory for a variable-length run skip count.

[Drawing 3] (A) - (D) is drawing showing the near multiplier to one example and memory configuration of a context model.

[Drawing 4] It is the explanatory view of one example of the significant memory configuration for the random access of a 16x16 sign block.

[Drawing 5] It is drawing showing the memory and the register which are used for the significant propagation pass for random access.

[Drawing 6] It is drawing showing the significant condition stored in the register for random access from memory.

[Drawing 7] It is the block diagram of one example of significant propagation pass logic.

[Drawing 8] It is drawing showing an example of the engine performance of one example of the context model on 4x4 blocks.

[Drawing 9] It is the explanatory view of one example of the configuration of the significant memory for sequential accesses of a 16x16 sign block.

[Drawing 10] It is the explanatory view of the memory used for significant propagation pass, and one example of a register.

[Drawing 11] It is the explanatory view of the memory for giving the suitable field for context model actuation, and the usage of a register.

[Drawing 12] It is the block diagram of one example of pass decision logic.

[Drawing 13] It is the block diagram of one example of double context generation logic.

[Drawing 14 A] It is the block diagram of one example of an initial context mold MQ encoder.

[Drawing 14 B] It is the explanatory view of one example of a typical decryption

implementation method.

[Drawing 15] It is the block diagram of one example of an anaphase context mold MQ encoder.

[Drawing 16 A] It is the explanatory view of the operating state of a comparison of a probability class index.

[Drawing 16 B] It is the block diagram of the multiplexer which determines MPS or LPS to each Qe value.

[Drawing 17] It is the explanatory view of the interval for a multiplex MPS decryption.

[Drawing 18] It is the explanatory view of one example of allocation of a juxtaposition sign block to 4:4:4 data.

[Drawing 19] It is the explanatory view of one example of allocation of a juxtaposition sign block to 4:2:2 data.

[Drawing 20] It is the explanatory view of other one example of allocation of a juxtaposition sign block to 4:1:1 data.

[Drawing 21] It is the block diagram of the memory of one example of the encoder which contains separately MQ encoder of a large number which have a relevance context model.

[Drawing 22 A] It is the explanatory view of the example which uses the bit plane of a number of memory restricted to each multiplier during coding.

[Drawing 22 B] It is the block diagram of one example of the control logic for controlling access to memory.

[Drawing 23] Instead of a counter, it is the explanatory view of the example which uses the memory of the small capacity for (variable-length VL) sign information.

[Drawing 24] It is the block diagram of one example of rectification.

[Drawing 25 A] It is the block diagram of one example of a low pass filter.

[Drawing 25 B] It is the block diagram of one example of a high-pass filter.

[Drawing 26 A] It is the block diagram of one example of a high-pass filter.

[Drawing 26 B] It is the block diagram of one example of a low pass filter.

[Drawing 27] It is the block diagram of other one example of rectification.

[Drawing 28] It is the block diagram of one example of inverse transformation.

[Drawing 29] It is the block diagram of one example of an encoder/decoder.

[Drawing 30] It is the explanatory view of an example of a 16-bit word which has both a run count and a skip count.

[Drawing 31] It is the explanatory view of 8x8 typical fields of the significant

condition bit which determines coding pass.

[Drawing 32] It is the block diagram of one example which determines pass logic.

[Description of Notations]

2901 Image Data Interface

2902 DC Level Shifter

2903 Wavelet Transform Machine

2905 Formation of Scalar Quantity Child / Reverse Quantizer

2906 PURIKODA

2907 Packet Header Treater

2908 Bit Modeling MQ Encoder

2911 Sign Memory

A, B Work-piece memory